Pre-print copy of the manuscript published by Elsevier
(available at: https://www.sciencedirect.com/science/article/pii/S016636152200118X)
identified by doi: 10.1016/j.compind.2022.103721

Reactive Synthesis of Software Robots in RPA from User Interface Logs

Simone Agostinelli, Marco Lupia, Andrea Marrella, Massimo Mecella
Sapienza Università di Roma, Rome, Italy

Abstract

Robotic Process Automation (RPA) is an emerging technology in the field of Business Process Management (BPM) that enables the automation of intensive repetitive tasks (or simply routines). RPA solutions access the user interface (UI) layer of software (SW) applications and provide a virtual workforce of SW robots that are able to mimic human keyboard and mouse interactions with a UI as if a real person was doing them. To take full advantage of this technology, organizations leverage the support of skilled human experts that preliminarily observe how routines are executed on the UI of the involved SW applications, and then implement the executable RPA scripts required to automate the routines enactment by SW robots on a target computer system. However, the current practice is time-consuming and error-prone, as it strongly relies on the ability of the human experts to correctly interpret the routines (and their variants) to automate. In this paper, to tackle this issue, we use a design science research method to develop an approach, called SmartRPA, which is able to interpret the UI logs keeping track of many routine executions, and to automatically synthesize SW robots that emulate the most suitable routine variant for any specific intermediate user input that is required during the routine execution. The approach is implemented as an open-source tool and evaluated with four non-functional requirements employing both syntectic and real-world data.

Email addresses: agostinelli@diag.uniroma1.it (Simone Agostinelli), lupia.1694700@studenti.uniroma1.it (Marco Lupia), marrella@diag.uniroma1.it (Andrea Marrella), mecella@diag.uniroma1.it (Massimo Mecella)

1. Introduction

Robotic Process Automation (RPA) is an emerging automation technology in the field of Business Process Management (BPM) that creates software (SW) robots to partially or fully automate rule-based and repetitive tasks (or simply *routines*) performed by human users in their applications' user interfaces (UIs) [1]. RPA is thought to provide the shortest route to business process (BP) automation by accessing only to the UI layer of IT systems rather than going deeply into the application code or databases sitting behind them [2].

In recent years, much progress has been made both in terms of research and technical development on RPA, resulting in many industry-specific deployments for industrial-oriented services [3, 4, 5, 6, 7, 8, 9]. Moreover, the market of RPA solutions has developed rapidly and today includes more than 50 vendors developing tools that provide SW robots with advanced functionalities for automating office tasks of different complexity [10]. Nonetheless, when considering state-of-the-art RPA technology, it becomes apparent that the current generation of RPA tools is driven by predefined rules and manual configurations made by expert users rather than automated techniques [11, 12, 13].

As investigated in [14, 15], in the early stages of the RPA life-cycle, it is required the support of skilled human experts that identify the candidate routines to automate by means of interviews and observation of workers conducting their daily work, and manually specify their conceptual and technical structure, often in form of *flowchart diagrams*, to enable the generation of executable scripts (also called *RPA scripts*) for the concrete enactment of SW robots at run-time. While this approach is proven effective to execute simple rules-based logic in situations where there is no room for interpretation, it becomes time-consuming and error-prone in presence of routines that are less predictable or require some level of human judgment. Indeed, the designer should have a global vision of all possible variants of the routines to define the appropriate behaviours of the SW robots, which becomes complicated when the number of variants increases. The

issue is that in case where the flowchart diagram does not contain a suitable response for a specific situation, e.g., because of an inaccurate modeling activity, then the associated RPA scripts would not properly reflect the behaviour of the potential routine variant, forcing SW robots to escalate to a human supervisor at run-time, in contrast with the RPA philosophy.

Although RPA is generally considered to be an easy to implement technology, in practice an in-depth knowledge is necessary to create reliable and scalable SW robots, in particular when intermediate user inputs are required to properly progress the execution of a routine. As a result, between 30% and 50% of initial RPA implementations are estimated to fail [16, 9]. Consequently, an approach that simplifies the realization of an RPA project, and in particular the generation of SW robots in presence of many routine variants, can be considered as a relevant artefact to investigate. This leads to the following research questions:

- **RQ1**: Which steps are required to make the generation of SW robots less dependent by the intervention of RPA human experts?
- **RQ2**: How can the detection of variants (and related variation points) in a routine be automatically achieved?
- **RQ3**: What is the effectiveness of employing an approach that synthesizes SW robots neglecting the (manual) specification stage of the routines' behaviour through flowchart models?

In answering these questions, in this paper – which extends our previous work [17] in several directions,[1] – we contribute to three recent challenges that were put forward in [11, 12, 13, 18], namely: **(C1)** the automated identification of the routine steps to robotize from a UI log, **(C2)** the automated detection of all the routine variants that require some user input to proceed with their execution, and **(C3)** the automated synthesis of executable RPA scripts for enacting SW

[1] For readability purposes, the details of the additional contributions with respect to our previous work [17] are explained at the end of Section 10.

robots at run-time. The result is an approach and an implemented tool, called SmartRPA, which is able to *(i)* interpret the UI logs recording the mouse/key events that happen on the UI of the SW applications involved in many routine executions, *(ii)* discover all the variants (and variation points) of the routine under observation, and *(iii)* automatically combine them into an executable RPA script, which can be *reactively* synthesized into a single SW robot.

Differently from the literature approaches to automated RPA scripts generation from UI logs (cf. Section 5), which enable to automate straightforward routines that have essentially no variance and do not require any human intervention, the SW robots generated by SmartRPA are obtained to handle the intermediate user inputs that are required during the routine execution, thus enabling to emulate the most suitable routine variant for any specific combination of user inputs as observed in the UI log. This makes the synthesis of SW robots performed by SmartRPA *reactive* to any user decision found during a routine execution. "Reactivity" highlights the fact that the behaviour of SW robots is determined immediately before their enactment, as it is driven by the specific user inputs required to execute the routine. Therefore, SmartRPA acknowledges the benefit of human involvement at multiple points of the routine execution, leveraging the "humans-in-the-loop" model for the automated execution of routines that are less static and require variable decisioning [13].

We structure the paper according to the activities suggested by Johannesson and Perjons in [19] for delivering a design science artifact. Specifically, Section 2 describes our research methodology. Section 3 presents the relevant background and preliminary concepts. Section 4 introduces a running example to explain the research challenges. Section 5 discusses the related work solutions that attempted to tackle the research challenges, with the aim to derive the technical requirements for the design of the SmartRPA approach, whose main steps are examined in Section 6. Section 7 outlines the algorithm for the automated detection of variation points from many routine executions. Section 8 analyzes the architecture and the technical aspects of the tool implementing the approach. Section 9 evaluates the robustness, feasibility, effectiveness and usability of the

approach and its implemented tool. Finally, Section 10 draws conclusions and trace future works.

2. Research Methodology

Our research methodology is inspired to the Design Science approach described by Johannesson and Perjons in [19]. The methodology is applied in four distinct sequential phases: problem formulation and objectives, requirements definition, design and development, and demonstration and evaluation. See Figure 1 for an overview.

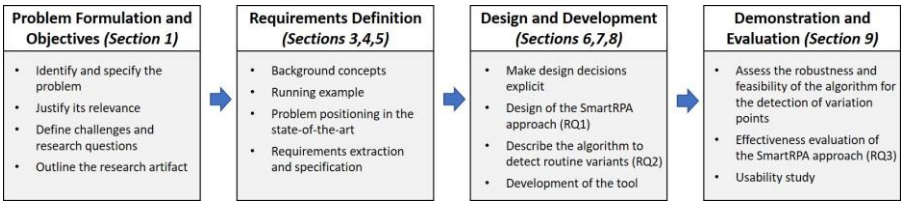


Figure 1: Research methodology based on Johannesson and Perjons [19]

Problem Formulation and Objectives. In this phase, which is tackled in Section 1, we first identify and specify the research problem to be tackled, i.e., the *reactive synthesis of SW robots in an automated way from UI logs*. Then, we justify its significance in the RPA field. The relevance of the problem is also supported by the presence of three related *research challenges*, i.e., **C1**, **C2** and **C3**, taken from previous works [11, 12, 13, 18]. Finally, we elaborate three main *research questions*, i.e., **RQ1**, **RQ2** and **RQ3**, for guiding our research towards the definition of an *artefact* to solve the problem. Such an artefact is represented by an approach and an implement tool, called SmartRPA, which is able to interpret the UI logs keeping track of many routine executions, and to automatically and reactively synthesize SW robots that emulate the most suitable routine variant for any specific intermediate user input that is required during the routine execution.

Requirements Definition. The second phase consists of eliciting the requirements on the outlined artefact. After providing the required background concepts on routines, SW robots and UI logs, we present a running example inspired by a real-life RPA use case to illustrate the relevance of the three challenges identified before. Then, we investigate the related work, including documented solutions to similar research challenges, with the aim to extract the technical requirements needed to support the design and development of SmartRPA.

Design and Development. Based on the analysis of the related work and the derived technical requirements, in the third phase we make design decisions explicit, discussing the SmartRPA approach and describing its stages to answer **RQ1**. Moreover, we present in detail a novel algorithm to automatically identify routine variants and variation points from UI logs, thus addressing **RQ2** and enabling a reactive synthesis of the SW robots. Lastly, we show the technical steps enacted to develop the SmartRPA approach as a real implemented tool.

Demonstration and Evaluation. In the fourth phase, with the aim to understand the general quality of SmartRPA to tackle the research challenges, we analyze four *non-functional requirements* on the artefact. Specifically, we first perform many synthetic experiments employing UI logs of increasing complexity to assess the *robustness* and *feasibility* of our approach to the identification of routine variants and variation points for the reactive synthesis of SW robots. Then, to answer **RQ3**, we perform a controlled experiment involving real users exploiting the use case of our running example to investigate the *effectiveness* of the SmartRPA approach when compared to a traditional model-based approach for the generation of SW robots. Finally, we quantify the *usability* of the UI provided by the tool implementing the SmartRPA approach.

3. Background and Preliminary Concepts

RPA can be seen as an evolution of screen scraping solutions [20], which sought to visualize screen display data from legacy applications (having no means for automated interfacing) to display such data using modern UIs. The

strength of RPA is that it does not replace existing applications or manipulate their code, but rather works with them in a way similar to a human user. In this section, we present the required background on RPA and preliminary concepts on routines, SW robots and UI logs needed to understand our approach.

3.1. Preliminaries on Routines and SW Robots

RPA moves around the concept of replacing routine work with automation. According to [21], a *routine* can be classified as a *structured process that reflects highly predictable and repetitive work with low flexibility requirements (i.e., the amount of variants to the expected process path is limited) and controlled interactions among process participants.*

While many overlapping definitions of RPA can be found in the research literature, in this paper we adopt the one proposed by Gartner in 2017 [22], which defines RPA as *a class of tools that enable users to specify routines involving [if, then, else] statements on structured data, rules, user interface interactions, and operations accessible via APIs. Such routines are encoded as scripts that are executed by SW robots, operated via control dashboards.*

Depending on how the control dashboard is exploited, it is possible to distinguish among *unattended* and *attended* SW robots.

- Unattended SW robots are able to fully automate routines without any intermediate human intervention. This happens when all execution paths are always the same independently by the specific inputs provided to the routine executions. For example, for the management of insurance claims (when claims are received in a structured form), unattended SW robots offer an efficient solution for their automated processing and validation. However, any variant to the expected behaviour of the routine is considered as an exception and, thus, redirected to human supervision.

- Attended SW robots work alongside humans, and are suitable for routines where some decisions or checks need to be made that require human judgement during the routines' execution. Therefore, attended SW robots

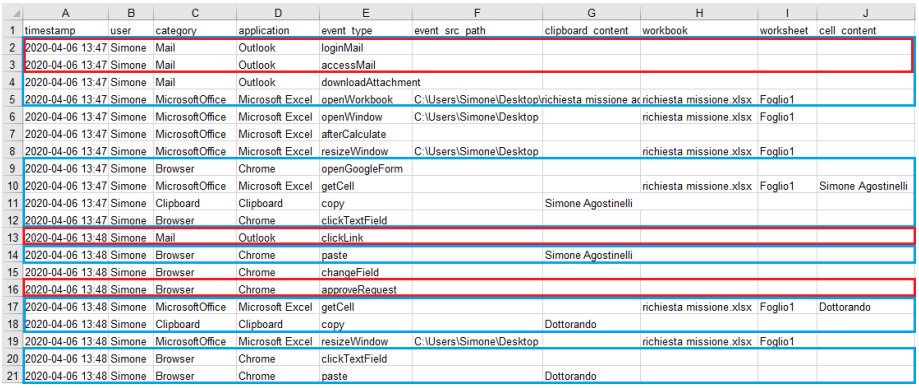
may require data from a user to properly progress the enactment of the routine. For example, a document-driven routine lends itself to attended automation because a human is entering information via a document, and different values of the provided information may potentially trigger the execution of different variants of the routine.

In a nutshell, unattended SW robots represent the simplest case of the attended perspective [18], since user inputs are not required for driving the routine's execution. On the other hand, attended SW robots are suitable in presence of *routine variants* recorded in the UI log. We define a routine variant as a *specific execution of a routine that differs from the other executions of the same routine by at least one event.* An *event* refers to the enactment of a user action (coupled with some execution data, like the name of the application where the action occurred, etc.) within a specific routine execution recorded in a UI log at a specific moment in time. The presence of different events in many routine executions *may potentially determine alternative behaviours* of the routine itself. This is particularly true when some events are triggered only by specific user inputs (and not by others) provided at the time of the routine execution. These events act as a *variation point* of the routine, where a user choice needs to be made between multiple possible variants. We will show an example of routine variants and variation points in the running example of Section 4.

3.2. Preliminaries on User Interface Logs

A UI log in its raw form consists of a timestamped sequence of events recorded during one user session.[2] Such events include all the user actions required to accomplish one or more relevant routines using the UI of one or many SW application/s. For instance, in Figure 2, we show a snapshot of a generic UI log captured using a dedicated action logger (that we will extensively discuss in Section 8) during the execution of two generic routines. The employed action

[2]We interpret a user session as a group of interactions that a single user takes within a given time frame on the UI of a specific computer system.



	A	B	C	D	E	F	G	H	I	J
1	timestamp	user	category	application	event_type	event_src_path	clipboard_content	workbook	worksheet	cell_content
2	2020-04-06 13:47	Simone	Mail	Outlook	loginMail					
3	2020-04-06 13:47	Simone	Mail	Outlook	accessMail					
4	2020-04-06 13:47	Simone	Mail	Outlook	downloadAttachment					
5	2020-04-06 13:47	Simone	MicrosoftOffice	Microsoft Excel	openWorkbook	C:\Users\Simone\Desktop\richiesta missione ac		richiesta missione.xlsx	Foglio1	
6	2020-04-06 13:47	Simone	MicrosoftOffice	Microsoft Excel	openWindow	C:\Users\Simone\Desktop		richiesta missione.xlsx	Foglio1	
7	2020-04-06 13:47	Simone	MicrosoftOffice	Microsoft Excel	afterCalculate					
8	2020-04-06 13:47	Simone	MicrosoftOffice	Microsoft Excel	resizeWindow	C:\Users\Simone\Desktop		richiesta missione.xlsx	Foglio1	
9	2020-04-06 13:47	Simone	Browser	Chrome	openGoogleForm					
10	2020-04-06 13:47	Simone	MicrosoftOffice	Microsoft Excel	getCell			richiesta missione.xlsx	Foglio1	Simone Agostinelli
11	2020-04-06 13:47	Simone	Clipboard	Clipboard	copy		Simone Agostinelli			
12	2020-04-06 13:47	Simone	Browser	Chrome	clickTextField					
13	2020-04-06 13:48	Simone	Mail	Outlook	clickLink					
14	2020-04-06 13:48	Simone	Browser	Chrome	paste		Simone Agostinelli			
15	2020-04-06 13:48	Simone	Browser	Chrome	changeField					
16	2020-04-06 13:48	Simone	Browser	Chrome	approveRequest					
17	2020-04-06 13:48	Simone	MicrosoftOffice	Microsoft Excel	getCell			richiesta missione.xlsx	Foglio1	Dottorando
18	2020-04-06 13:48	Simone	Clipboard	Clipboard	copy		Dottorando			
19	2020-04-06 13:48	Simone	MicrosoftOffice	Microsoft Excel	resizeWindow	C:\Users\Simone\Desktop		richiesta missione.xlsx	Foglio1	
20	2020-04-06 13:48	Simone	Browser	Chrome	clickTextField					
21	2020-04-06 13:48	Simone	Browser	Chrome	paste		Dottorando			

Figure 2: Snapshot of a UI log captured during the executions of two generic routines

logger enables to record the events happened on the UI, enriched with several data fields describing their payload. For a given event, such fields are useful to keep track the name and the timestamp of the user action performed on the UI, the involved SW application, the human/SW resource that performed the action, etc. In general, events recorded by different SW applications are characterized by different data fields. For example, the events generated by a spreadsheet (e.g., an Excel spreadsheet) contain information such as spreadsheet name and position of the involved cell or range of cells, while Web-based events are characterized by the name of the corresponding HTML web page.

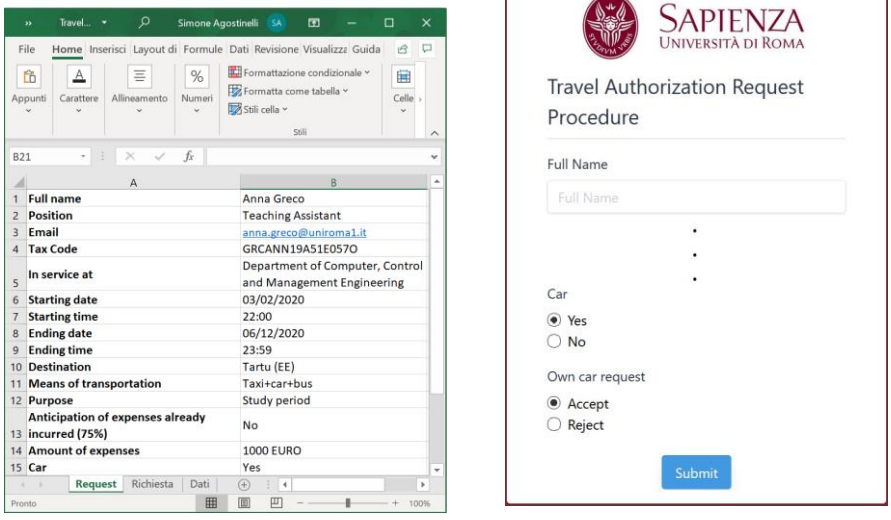
As shown in Figure 2, a UI log may contain multiple and interleaved executions of one/many routine/s (cf. the blue/red boxes that group the events belonging to two generic different routines), as well as redundant behavior and noise. We consider as *redundant* any event that is unnecessarily repeated during the execution of a routine, e.g., a text value that is first pasted in a wrong field by mistake and then is moved in the right place through a corrective action on the UI. On the other hand, we consider as *noise* all those events that do not contribute to the achievement of any routine target, e.g., a window that is resized. In Figure 2, the sequences of events that are not surrounded by a blue/red box could be safely labeled as noise.

We conclude this section by introducing the concepts of *routine trace*, which

represents an execution instance of a routine within a UI log (e.g., the events surrounded by a blue/red box belong to two routine traces related to two different routines), and of *routine-based log* as a container that stores the routine traces extracted by a UI log and related to a specific routine. Conceptually speaking, routine-based logs resemble event logs in Process Mining [23].

4. Running Example

In this section, we describe a RPA use case inspired by a real-life scenario at Department of Computer, Control and Management Engineering (DIAG) of Sapienza Università di Roma. The scenario concerns the filling of the travel authorization request form made by professors, researchers and PhD students of DIAG for travel requiring prior approval. The request applicant must fill a well-structured Excel spreadsheet (cf. Figure 3(a)) providing some personal information, such as her/his bio-data and the email address, together with further information related to the travel, including the destination, the starting/end-



(a) Excel spreadsheet (b) Google form

Figure 3: UIs involved in the running example

ing date/time, the means of transport to be used, the travel purpose, and the envisioned amount of travel expenses, associated with the possibility to request an anticipation of the expenses already incurred (e.g., to request in advance a visa). When ready, the spreadsheet is sent via email to an employee of the Administration Office of DIAG, which is in charge of approving and elaborating the request. The delivery of the travel authorization request in the email inbox of an employee triggers the starting of the routine procedure described below.

For each row in the spreadsheet, the employee manually copies every cell in that row and pastes that into the corresponding text field in a dedicated Google form (cf. Figure 3(b)), accessible just by the Administration staff. In addition, if the request applicant declares that s/he would like to use her/his personal car as one of the means of transport for the travel (by filling the dedicated row labeled with "Car" in the spreadsheet), then the employee has to activate the request on the Google form (in this case, a dialog box labeled "Own car request" appears on the UI, cf. Figure 3(b)) and then accept (only in this case a special insurance is automatically activated for the part of the travel that will be performed with the car) or reject the personal car request. When the data transfer for a given travel authorization request has been completed, the employee presses the "Submit" button to submit the data into an internal database. Once the form is submitted, a confirmation email is sent automatically to the applicant.

The above routine procedure (in the following, we will denote it as $R_{example}$) is usually performed manually, it is time consuming (as it must be repeated for any new travel request) and prone to errors. For the sake of understandability, we show in Figure 4 the flowchart model of $R_{example}$, represented through the ISO/IEC 19510:2013 standard BPMN notation.

Analyzing the BPMN model in Figure 4, it becomes clear that a proper execution of $R_{example}$ requires a path on the UI made by the user actions reported in Table 1.[3] Note that actions openWorkbook and openGoogleForm can be per-

[3]Note that the user actions recorded in a UI log can have a finer granularity than the high-level ones used here just with the purpose of describing the routine's behaviour.

formed in any order, and the sequence of actions ⟨ getCell, copy, clickTextField, paste⟩ can be repeated for any travel information to be moved from the Excel spreadsheet to the Google form. Finally, in case of a car request to be evaluated (activateCarRequest), the execution of accept or reject is exclusive.

Depending on the order/choice/number of repetitions of the above user actions, many different variants of $R_{example}$ can be potentially emulated. However, the behaviour implied by $R_{example}$ semantically changes only after the enactment of the action activateCarRequest, which requires an explicit user decision between the possibility of *accepting* or *rejecting* the personal car request. Therefore, the actions accept and reject represent a *variation point* of the routine, that forks its execution flow into two well distinguished exclusive branches.

The majority of commercial RPA tools enable RPA user experts to tag the variation points directly in the flowchart model. That is, $R_{example}$ can be modeled and properly emulated by a SW robot if an in-depth knowledge of

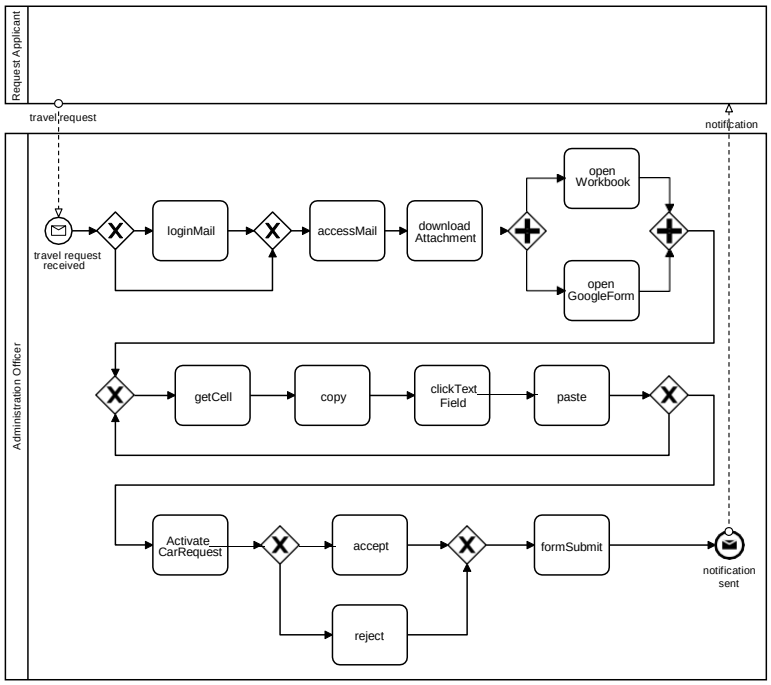


Figure 4: Flowchart model of $R_{example}$ describing its main steps

Name	Description
loginMail	to access the client mail
accessMail	to access the specific email with the travel request
downloadAttachment	to download the Excel file including the travel request
openWorkbook	to open the Excel spreadsheet
openGoogleForm	to access the Google Form to be filled
getExcelCell	to select the cell in the i-th row of the Excel spreadsheet
copy	to copy the content of the selected cell
clickGoogleFormTextField	to select the specific text field of the Google form where the content of the cell should be pasted
paste	to paste the content of the cell into the corresponding text field of the Google form
activateCarRequest	to activate in the Google form the dialog box for approving or rejecting the car request
accept	to press the button on the Google form that approves the request
reject	to press the button on the Google form that rejects the request
formSubmit	to press the button to finally submit the Google form to the internal database

Table 1: User actions required to execute $R_{example}$

the anatomy and working of the routine is available during the modeling task. But without such a knowledge, which is based on careful observation sessions of human users that perform routine tasks in their computer systems, it becomes extremely complex both to identify the candidate steps of the routine to specify in the flowchart model (cf. **C1**) and the detection of those variants that would require some user inputs to proceed with their execution (cf. **C2**). In a nutshell, the ability of commercial RPA tools to emulate all the possible behaviours of the routine depends by the correctness of the modeling task, without which is not possible automatically generate the executable RPA scripts to be embedded into the SW robots (cf. **C3**).

In this direction, in the next section we first present the relevant approaches from the research literature that are able to mitigate the above challenges by skipping the modeling activity of the flowchart diagram. Then, in an attempt to fully address them, we derive a set of technical requirements to realize our SmartRPA approach.

5. Related Work Analysis and Requirements Specification

The research literature proposes many approaches that are targeted to automatically discover and implement the behaviour of SW robots by interpreting the working of the routines stored into previously recorded UI logs.

Towards the addressing of **C1**, the works [24, 25] provide an approach, coupled with an implemented tool, that leverages process mining techniques to *(i)* keep track of UI actions performed within Excel and Google Chrome into an event log, and *(ii)* extract the fragments of a routine that can be eventually automated by a third-party RPA tool. Similarly, in [26] it is presented the *Desktop Activity Mining* tool, which is able to record the user actions performed during an office-based routine task on a UI, and to discover a process model describing the behaviour of such routine. Note that the proposed tool is based on recording the mouse click coordinates on the screen and store them in a dedicated UI log, and thus it can not replicate the same user's observed behavior performed in different computer systems, lacking portability.

Even if the works [24, 25, 26] do not tackle the issue of synthesizing executable RPA scripts from the identified candidate routines, they had the vision that the behaviour of a routine can be inferred by observing and interpreting the footprint of the routine itself from a UI log that keeps track of its user actions. This directly leads to three technical requirements that need to be met to tackle **C1**:

Req_1 - Recording UI Logs: A feature to record the low-level user actions executed during one (or many) routine(s) enactment on the UI in form of a UI log is strongly needed to keep track of its behaviour.

Req_2 - Extraction of Routine-based Logs from a UI Log: A UI log may contain interleaved executions of one/many routine/s. As the target is to reason on the behaviour of a single routine per time, it is needed to preprocess the UI log to: (1) identify which user actions contribute to which routines inside the UI log; (2) organize such actions into well-bounded routine traces and (3) store them into a dedicated routine-based log.

Req$_3$ - **Events Abstraction**: A routine-based log is characterized by low-level user actions, and thus may contain noise and redundant actions (cf. Section 3 for their definition) that must be filtered out from the log itself.

With the aim to tackle **C2**, in [27], the authors propose a self-learning approach to automatically detect high-level RPA-rules from historical low-level behaviour logs. An if-then-else deduction logic is used to infer rules from behaviour logs by learning relations between the different routines performed in the past. Then, such rules are employed to facilitate the SW robots instantiation. A similar approach is adopted in [28], where the *FlashExtract* framework is presented. FlashExtract allows to extract relevant data from semi-structured documents using input-output examples, from which one can derive the relations underlying the working of a routine. Finally, in [29] the authors identify repetitive edits to text documents by keeping track of a graph of edits and suggest automation rules for SW robots.

The above works have provided a relevant contribution for the semi-automatically detection of variation points of a routine, with the aim to support the manual development of SW code by RPA expert users. In the direction of realizing a fully automated approach to the detection of routine variants and variation points, we can derive the following requirement:

Req$_4$ - **Automated Detection of Variation Points**: An algorithm that is able to automatically detect the variation points of a routine from a routine-based log is required to reactively generate SW robots that emulate properly the behaviour of the routine.

Concerning **C3**, the literature proposes only a relevant solution, called Robidium [30], that tackles this challenge. Robidium is an approach and an open-source tool that enables to generate executable scripts (by only interpreting UI logs) that can be enacted by the commercial RPA tool UI Path.[4]

[4] www.uipath.com

The main feature of Robidium is that it automates only the most frequent routine variant among the ones discovered in the UI log. This because Robidium synthesizes RPA scripts that do not require intermediate user inputs during their execution, i.e., it is focused on the generation of unattended SW robots (cf. Section 2). On the other hand, in order to synthesize attended SW robots, the following requirement is needed:

Req_5 - Automated and Reactive Generation of SW Robots: A solution that is able to automatically and reactively synthesize RPA scripts is required for the generation of attended SW robots able to enact the most suitable routine variant depending on the specific input conditions at hand.

It is worth to notice that another group of approaches exists towards SW robots automation, which focuses on learning the structure of a routine from natural language descriptions of the procedure underlying the routine itself. In this direction, the work [31] defines a new grammar for complex workflows with chaining machine-executable meaning representations for semantic parsing. In [32], the authors provide an approach to learn activities from text documents employing supervised machine learning techniques such as feature extraction and support vector machine training. Similarly, in [33] the authors adopt a deep learning approach based on Long Short-Term Memory (LSTM) recurrent neural networks to learn the relationship between activities of a routine task. The above works assume the availability of textual documentation of suitable quality and completeness at the outset, and neglect the fact that users can perform steps in a routine that are not fully documented to deal with variations and exceptions. This may potentially led to imprecise results in the description of the routine's anatomy. Therefore, these works seems to be particularly suitable to discover the desired structure of a routine, in contrast with the observed one, like happens in all the log-based approaches discussed so far.

Finally, a third group of approaches exist that aim to eliminate human-dependent training [34, 35]. They rely on probabilistic and machine learning al-

gorithms to automatically train SW robots, so that any manual effort is avoided. These approaches are currently the least mature if compared with the others discussed above, but potentially with the best promises for realizing fully automated intelligent RPA approaches.

6. Design of the SmartRPA Approach

From a methodological perspective, SmartRPA has been conceptualized and designed towards addressing the five technical requirements discussed in Section 5. In addition, the approach underlying SmartRPA takes inspiration from the RPM (Robotic Process Mining) framework presented by Leno et al. in [18]. RPM aims to support analysts to produce executable specifications of routines, in form of SW robots, interpreting the routine executions stored in a UI log. Specifically, RPM envisions a pipeline of three main stages that consist of: *(i)* collecting and pre-processing UI logs corresponding to executions of one or more routine executions; *(ii)* identifying and discovering candidate routines to be automated with RPA tools; and *(iii)* synthesizing executable RPA scripts. Robidium [30] is a concrete example of how to realize the RPM approach.

To address the technical requirements, SmartRPA incorporates the three main stages of the RPM framework within a larger approach that includes five operational steps to be applied in sequence: *(i)* Log Recording, *(ii)* Log Processing, *(iii)* Event Abstraction, *(iv)* Process Discovery, and *(v)* Script Generation, as shown in Figure 5. Note that such methodological steps are useful not only to tackle the technical requirements, but also serve as our answer to **RQ1**.

Figure 5: Overview of the SmartRPA approach

Log Recording. SmartRPA belongs to the category of those approaches that learn how to automate routines "by examples". Therefore, a UI log that keeps

track of the low-level user actions generated during the interaction with the UIs of multiple SW applications within the execution of a routine is required to derive its structure. To this end, a training session in which several users perform the same routine to be automated is necessary to properly record the specific UI actions involved in its execution. While there exist many monitoring and recording solution in the Human-Computer Interaction field [36] that keep track of the actions that a user is doing on the screen of a computer system (recorded as mouse click coordinates) during a controlled experiment, in SmartRPA we need dedicated recording features to produce a raw UI log corresponding to many executions of the same routine during a pre-defined period of time (cf. **Req**$_1$). At the end of the training session, the outcome of this step will consist of as many UI logs as are the users that performed the routine under analysis from the start to the end.

Log Processing. It comes into play to pre-process the recorded UI logs and make them suitable for being properly interpreted. Since any UI log obtained from the previous step keeps track of single, independent execution of the observed routine, a merging activity is needed to combine them into a single, larger UI log. In a nutshell, the content of any recorded UI log obtained after a training session will be considered as a single trace of the (larger) UI log being generated. Thus, if compared with the description of UI logs performed in Section 2, we can say that SmartRPA enables to interpret UI logs that are *routine-based*, i.e., logs that can be already considered as *well segmented*, since the enactment of any training session will be represented by a specific routine trace in the log (cf. **Req**$_2$). Of course, this does not prevent the presence of noise and redundant user actions in the recorded routine traces, whose presence will be reduced in the next steps.

Event Abstraction. This step is targeted to convert the routine-based UI log (that will be later employed to generate the executable RPA scripts), which contains the low-level user actions recorded during the interaction with the UI, into a high-level version of it. Such high-level version can be used for

diagnostic and analysis purposes by expert RPA analysts to: *(i)* filter out noise, i.e., irrelevant events for the routine execution. For instance, applications related to the operating system such as *Windows Updater* may start automatically while the UI log is being recorded, and they may dirty the recording phase of the users during their training session, thus they need to be filtered out; *(ii)* group similar events, mitigating noise (cf. $\mathbf{Req}_3$). Moreover, the high-level routine-based UI log will be used to derive the flowchart representing the abstract workflow describing the routine behaviour, employing dedicated high-level descriptive labels to represent the high-level activities. We notice that the Event Abstraction step is an addition with respect to the RPM framework, which instead focuses on discovering the anatomy of a routine only for execution purposes.

Process Discovery. This step has a twofold objective:

- It takes in input the high-level routine-based log generated by the Event Abstraction component to derive the workflow describing the users' observed behavior in the UI. This workflow can be analyzed by an RPA analyst to look at the high-level structure of the routine under analysis.
- Moreover, the knowledge of the workflow underlying the routine, coupled with the low-level version of the routine-based UI log, will be used to support the detection of the most suitable routine variant according to the intermediate user inputs observed into the UI log, and its encoding into a SW robot. In Section 7, it is reported a detailed discussion about the algorithm implemented to the identification of the routine variants and the related variation points (cf. $\mathbf{Req}_4$), necessary to obtain a reactive synthesis of the SW robot.

It is worth to notice that in the RPM framework the Process Discovery step shown above is (in part) realized through the second stage of the framework, targeted to the identification and discovering of the candidate routines to be automated. In SmartRPA, the knowledge about which routine has to be automated is already embedded into the UI logs obtained by performing the training

session at the outset. Moreover, during the Event Abstraction step, the routine is further cleaned from noise and redundancies, keeping just the events in the UI that can contribute to the routine automation.

Script Generation. First of all, this step allows a RPA analyst to personalize the values stored in the events of the most suitable routine variant detected in the previous step, before the generation and enactment of the SW robot. Finally, taking into account the edits made, SmartRPA can finally generate the required executable RPA script to run the SW robot that emulates the most suitable routine execution on the UI, by scanning the recorded low-level events stored in the routine-based UI log and converting them into executable pieces of SW code (cf. $\mathbf{Req}_5$).

7. An Algorithm for the Automated Detection of Variation Points of a Routine

To properly address **RQ2**, in this section we present an algorithm to identify different variation points of a routine by inspecting multiple executions of the routine itself inside the low-level routine-based log obtained as the outcome of the Event Abstraction step. A *variation point* is a point in the routine execution where a user choice needs to be made between multiple possible variants (cf. Section 3). The identification of variation points is fundamental to synthesize SW robots that emulate the most suitable routine variants in relation to the intermediate user inputs provided during the routine enactment. To be more specific, Algorithm 1 takes in input the low-level routine-based log and builds in output a new routine-based log that clearly categorizes the user actions that contribute to the identification of a new routine variant, distinguishing them from the (other) actions that are common to any routine trace recorded in the UI log. In the following, we discuss the main steps of Algorithm 1 relying on the running example explained in Section 4.

Aligning (`line 2`). The first step of the algorithm consists of aligning the different executions recorded in the routine-based log to make them more similar

Algorithm 1 Reactive Synthesis of SW Robots

```
1: procedure generateRoutineVariant(df: DataFrame)
2:     df ← align rows                                   ▷ Aligning step
3:     df["duplicated"] ← mark duplicated rows           ▷ Marking step
4: resultDF = None
5: previousDecidedDF = None
6:     groups ← group rows of df                         ▷ Grouping step
7:     for groupDF in groups do                          ▷ Iterating step
8:         if groupDF.duplicated == True then            ▷ no decision to take
9:             rows ← get the rows of the first trace from groupDF
10:            resultDF.append(rows)
11:        else                                          ▷ variation point
12:            if previousDecidedDF then
13:                IDs ← case IDs of traces compliant with previousDecidedDF
14:                filteredDF ← rows with case ID in IDs from groupDF
15:                decisionDF ← remove redundant rows from filteredDF
16:            else
17:                decisionDF ← remove redundant rows from groupDF
18:            end if
19:            decisionDialog ← show decision dialog built from decisionDF
20:            decidedDF ← rows of decisionDF selected in decisionDialog
21:            resultDF.append(decidedDF)  ▷ append rows from decidedDF
22:            previousDecidedDF ← decidedDF             ▷ save current decision
23:        end if
24: end for
25: return resultDF
26: end procedure
```

from each other, when possible. This means, on the one hand, removing user actions that are irrelevant for the execution of the SW robot, such as special URLs like about:blank or chrome://newtab/ or low-level events such as

enableBrowserExtension or afterCalculate. And, on the other hand, identifying and moving in the same point of any trace of the log those sequences of events underlying exactly the same behaviour in different traces (e.g., copy-/paste activities from a specific cell to specific text field) but originally located in different points among the various traces.

Marking (`line 3`). To enact the most suitable routine variant, it is important to identify which variation points need to be considered within many routine executions. To this end, the very first step consists of marking as "duplicated" those rows that underly exactly the same event performed on the UI in different routine traces of the log. In a nutshell, the *it-h* row of a routine trace is considered as duplicated if it includes an event that is exactly the same in the *it-h* row of all the other routine traces. We evaluate two events as identical if the following data fields have the same value for the event in the *it-h* row in all the recorded routine traces:

- *category*: represents the category of the user action, e.g., *Browser*, *OperatingSystem*, *Clipboard*, and *MicrosoftOffice*;
- *application*: name of the application where the user action occurred, e.g., Google Chrome, Microsoft Excel, etc.;
- *concept:name*: name of the user action recorded by the Action Logger component;
- *event src path*: source path in the operating system related to a user action. It could indicate the path of a file or folder opened, modified, created or deleted. It could also denote the path of an executable program that has been opened or closed;
- *event dest path*: destination path in the operating system related to a user action. If a file or folder is renamed, the new path name is present in this column;

- *browser url hostname*: hostname of the url recorded within a routine-based log. Two rows could have different URLs but the same hostname (e.g., *www.uniroma1.it/students* and *www.uniroma1.it/contacts* both have *uniroma1.it* as hostname);
- *xpath*: *XML Path Language* is a query language for selecting nodes in a page. It is used to uniquely identify a HTML element in a webpage.

	case:concept:name	category	application	concept:name	browser_url	tag_value	duplicated
0	1109155433758000	MicrosoftOffice	Microsoft Excel	openWorkbook			True
1	1109155433758000	Browser	Chrome	startDownload	about:blank		True
2	1109155433758000	Clipboard	Clipboard	copy			True
3	1109155433758000	MicrosoftOffice	Microsoft Excel	editCell			True
4	1109155433758000	Browser	Chrome	changeField	https://bpm-diag.github.io/form/	Vanessa Costa	True
5	1109155433758000	Browser	Chrome	clickRadioButton	https://bpm-diag.github.io/form/	car_no	False
6	1109155433758000	Browser	Chrome	clickButton	https://bpm-diag.github.io/form/		True
7	1109155433758000	Browser	Chrome	submit	https://bpm-diag.github.io/form/		True
8	1109155433758000	Browser	Chrome	formSubmit	https://bpm-diag.github.io/form/		True
9	1109155658364000	MicrosoftOffice	Microsoft Excel	openWorkbook			True
10	1109155658364000	Browser	Chrome	startDownload	about:blank		True
11	1109155658364000	MicrosoftOffice	Microsoft Excel	editCell			True
12	1109155658364000	Clipboard	Clipboard	copy			True
13	1109155658364000	Browser	Chrome	changeField	https://bpm-diag.github.io/form/	Vanessa Marino	True
14	1109155658364000	Browser	Chrome	clickRadioButton	https://bpm-diag.github.io/form/	car_yes	False
15	1109155658364000	Browser	Chrome	clickRadioButton	https://bpm-diag.github.io/form/	car_accept	False
16	1109155658364000	Browser	Chrome	clickButton	https://bpm-diag.github.io/form/		True
17	1109155658364000	Browser	Chrome	submit	https://bpm-diag.github.io/form/		True
18	1109155658364000	Browser	Chrome	formSubmit	https://bpm-diag.github.io/form/		True
19	1111145627144000	MicrosoftOffice	Microsoft Excel	openWorkbook			True
20	1111145627144000	Browser	Chrome	startDownload	about:blank		True
21	1111145627144000	MicrosoftOffice	Microsoft Excel	editCell			True
22	1111145627144000	Clipboard	Clipboard	copy			True
23	1111145627144000	Browser	Chrome	changeField	https://bpm-diag.github.io/form/	Paola Valentini	True
24	1111145627144000	Browser	Chrome	clickRadioButton	https://bpm-diag.github.io/form/	car_yes	False
25	1111145627144000	Browser	Chrome	clickRadioButton	https://bpm-diag.github.io/form/	car_reject	False
26	1111145627144000	Browser	Chrome	clickButton	https://bpm-diag.github.io/form/		True
27	1111145627144000	Browser	Chrome	submit	https://bpm-diag.github.io/form/		True
28	1111145627144000	Browser	Chrome	formSubmit	https://bpm-diag.github.io/form/		True

Figure 6: Excerpt of the routine-based log describing 3 out of 50 routine traces of $R_{example}$

Only the above subset of data fields associated to an event in the UI log is evaluated to the detection of duplicate rows, because some data fields are always different among the several executions of a routine. For example, the *case ID* of a routine trace or its *timestamp* are unique, and would always lead to false results

if they were considered, because all the rows would always result different (i.e., not duplicated), leading to the identification of many wrong variation points.

To sum up, if an event appears in all the routine traces of a routine-based log and the rows associated to that event have the same values for all the data fields discussed above, it means the users executed always the same user action on the UI at a specific point of the routine execution during their training session. As a consequence, such "duplicated" events would not lead to any variation point of the routine.

On the other hand, if there exists at least a user that performed an action on the UI in the *i-th* step of a routine trace that differs (according to the data fields listed above) from the actions performed at the same *i-th* step of the other routine executions, then it means that the associated event only appears in certain routine traces and not in others, thus identifying a variation point. As a consequence, we mark as "not duplicated" all the *i-th* rows of any routine trace under analysis. From a technical point of view, a new column accepting boolean values called *duplicated* is added to the routine-based log. Figure 6 shows a fragment of the user actions belonging to 3 different routine traces of $R_{example}$ identifying a variation point that leads to three different routine variants. For each row, if the corresponding *duplicated* field is set to *True*, it means the user action associated to that row is present in all the routine traces of the routine-based log and the values contained in the aforementioned data fields are the same across all the routine traces. Otherwise, *duplicated* is set to *False*.

Grouping (`line 6`). Once all the rows of the low-level routine-based log have been marked, for each routine trace, the algorithm evaluates them sequentially (following the timestamped ordering of events in the trace) and creates different *groups* of events according to the following conditions:

1. all the sequential rows having the column *duplicated* set to False, that *precede* (but are *not preceded* by) a row with the column *duplicated* set to True, are added to a new group. It is worth noticing this condition is

satisfied only when a routine trace starts with a sequence of rows having the column *duplicated* set to False;

2. all the sequential rows having the column *duplicated* set to False that *precede* a row with the column *duplicated* set to True (and for which condition 1 does not hold), are added to a new group;

3. all the sequential rows having the column *duplicated* set to True are added to a new group.

In a nutshell, a new group of events will be created for any different sequence of events in a routine trace having the column *duplicated* set to True or False. When the above 3 steps have been applied for any routine trace in the UI log, the *i-th* groups of each trace will be merged in a larger *i-th* group associated to the UI log. To better understand the rationale of the grouping procedure, let's analyze the routine-based log depicted in Figure 6:

- the sequence of rows [0,4] has the column *duplicated* equals to True, since the user actions associated to that rows are present in all the 3 recorded routine traces, and the values contained in the aforementioned columns are the same across all the routine traces. They violate grouping conditions 1 and 2 but satisfy condition 3. Then, they are added to a group, namely A (cf. Figure 7);

- row 5 has the column *duplicated* equals to False and it follows grouping condition 2, thus it is added to a new group, namely B (cf. Figure 7);

- the sequence of rows [6,8] has the column *duplicated* equals to True for the same reason of the first item. It violates grouping conditions 1 and 2 but satisfies condition 3, thus it is added to a new group, namely C (cf. Figure 7).

The same reasoning can be performed for the other routine traces of the routine-based log. Indeed:

- the sequences of rows [9,13] and [19,23] are added to group A;

- the sequences of rows [14,15] and [24,25] are added to group B;
- the sequences of rows [16,18] and [26,28] are added to group C.

A	case:concept:name	category	application	concept:name	browser_url	tag_value	duplicated
0	1109155433758000	MicrosoftOffice	Microsoft Excel	openWorkbook			True
1	1109155433758000	Browser	Chrome	startDownload	about:blank		True
2	1109155433758000	Clipboard	Clipboard	copy			True
3	1109155433758000	MicrosoftOffice	Microsoft Excel	editCell			True
4	1109155433758000	Browser	Chrome	changeField	https://bpm-diag.github.io/form/	Vanessa Costa	True
9	1109155658364000	MicrosoftOffice	Microsoft Excel	openWorkbook			True
10	1109155658364000	Browser	Chrome	startDownload	about:blank		True
11	1109155658364000	MicrosoftOffice	Microsoft Excel	editCell			True
12	1109155658364000	Clipboard	Clipboard	copy			True
13	1109155658364000	Browser	Chrome	changeField	https://bpm-diag.github.io/form/	Vanessa Marino	True
19	1111145627144000	MicrosoftOffice	Microsoft Excel	openWorkbook			True
20	1111145627144000	Browser	Chrome	startDownload	about:blank		True
21	1111145627144000	MicrosoftOffice	Microsoft Excel	editCell			True
22	1111145627144000	Clipboard	Clipboard	copy			True
23	1111145627144000	Browser	Chrome	changeField	https://bpm-diag.github.io/form/	Paola Valentini	True

B	case:concept:name	category	application	concept:name	browser_url	tag_value	duplicated
5	1109155433758000	Browser	Chrome	clickRadioButton	https://bpm-diag.github.io/form/	car_no	False
14	1109155658364000	Browser	Chrome	clickRadioButton	https://bpm-diag.github.io/form/	car_yes	False
15	1109155658364000	Browser	Chrome	clickRadioButton	https://bpm-diag.github.io/form/	car_accept	False
24	1111145627144000	Browser	Chrome	clickRadioButton	https://bpm-diag.github.io/form/	car_yes	False
25	1111145627144000	Browser	Chrome	clickRadioButton	https://bpm-diag.github.io/form/	car_reject	False

C	case:concept:name	category	application	concept:name	browser_url	tag_value	duplicated
6	1109155433758000	Browser	Chrome	clickButton	https://bpm-diag.github.io/form/		True
7	1109155433758000	Browser	Chrome	submit	https://bpm-diag.github.io/form/		True
8	1109155433758000	Browser	Chrome	formSubmit	https://bpm-diag.github.io/form/		True
16	1109155658364000	Browser	Chrome	clickButton	https://bpm-diag.github.io/form/		True
17	1109155658364000	Browser	Chrome	submit	https://bpm-diag.github.io/form/		True
18	1109155658364000	Browser	Chrome	formSubmit	https://bpm-diag.github.io/form/		True
26	1111145627144000	Browser	Chrome	clickButton	https://bpm-diag.github.io/form/		True
27	1111145627144000	Browser	Chrome	submit	https://bpm-diag.github.io/form/		True
28	1111145627144000	Browser	Chrome	formSubmit	https://bpm-diag.github.io/form/		True

Figure 7: Grouping rows of the low-level routine-based log

Iterating groups (`lines 7-25`). Once all groups have been identified, they are analyzed one by one in a cycle. For each identified group, namely *groupDF* (line 7), if the corresponding column *duplicated* is True for all the rows contained in

it (line 8), it means that all the routine traces in that group contain the same user actions. In this case, since that group does not identify a variation point, the rows of the routine trace appearing first in the group (line 9) are directly added to *resultDF* (line 10). Note that choosing the rows of another trace rather than the first one recorded in the group would lead to the same effect. Conversely, if the column *duplicated* is False, it means that we have detected a variation point to be considered.

When a variation point is detected by the algorithm, it is important to ensure that it is consistent with respect to the routine path that was executed until that point. Indeed, during each iteration, the rows associated with the previous decided user actions (decided user action are those actions selected when a variation point is identified) are saved in *previousDecidedDF* (line 22). A custom routine-based log called *decisionDF* is created to store the rows of the current decision about which user actions enact in presence of a variation point. In the first cycle iteration, no decision has been made, so *decisionDF* is generated only from the rows of the group that is currently processed (line 17). In the subsequent iterations, the current group along with the previous decision are taken into account to find the next possible variation point. The case IDs of the routine traces that have rows in common with the previous decision are selected (line 13). Then, the rows of the routine traces having those case IDs are picked from the current group *groupDF* and stored in *filteredDF* (line 14). This step ensures that the next possible variation point is in the routine path that starts from the previous detected variation point.

Finally, *decisionDF* is generated from the rows in *filteredDF* (line 15). In this step, redundant rows are filtered out. Two or more rows of different routine traces are considered as *redundant* if the associated user actions store the same values for the columns mentioned in the step "*Marking*". At this point, the user can choose which user actions to enact in the range of any identified variation point by means of a custom dialog (e.g., see Figure 8) that is launched just before the script generation step of the approach.

The custom dialog displays data from *decisionDF* (line 19), which is used

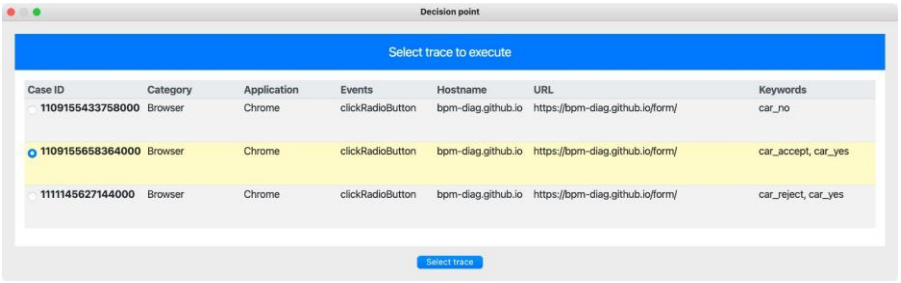


Figure 8: Custom dialog window to enact the user actions of 3 routine traces of $R_{example}$ in presence of a variation point

to bound together all the rows of a routine trace into a single line when a variation point is detected. To better understand this, consider the group of rows with column *duplicated* equals to False in Figure 7 containing 5 rows belonging to 3 different routine traces. The user has to decide which user actions of which routine trace enact, so these 5 rows are grouped together by their ID, and the names of the user actions related to each routine trace are flattened into a single line. Indeed, the dialog in Figure 8 shows a variation point that contains 3 different user inputs that led to 3 different execution variants of $R_{example}$: each line represents a routine trace because it has a unique case ID, and all the user actions names of each routine trace are flattened into the same line.

Once the user decides which user actions to execute (line 20), the corresponding rows are appended to the output routine-based log *resultDF* (line 21). It contains all the rows related to user decisions as well as rows with column *duplicated* equals to True (common to every routine trace). Note that *resultDF* will be the input of the Script Generation step of the SmartRPA approach.

8. Architecture and Development of SmartRPA

Starting from the approach outlined in Figure 5, the architecture of SmartRPA integrates five main SW components developed in Python that enable the reactive synthesis of SW robots according to the to intermediate user inputs recorded in the UI logs, thus emulating the most suitable routine variant

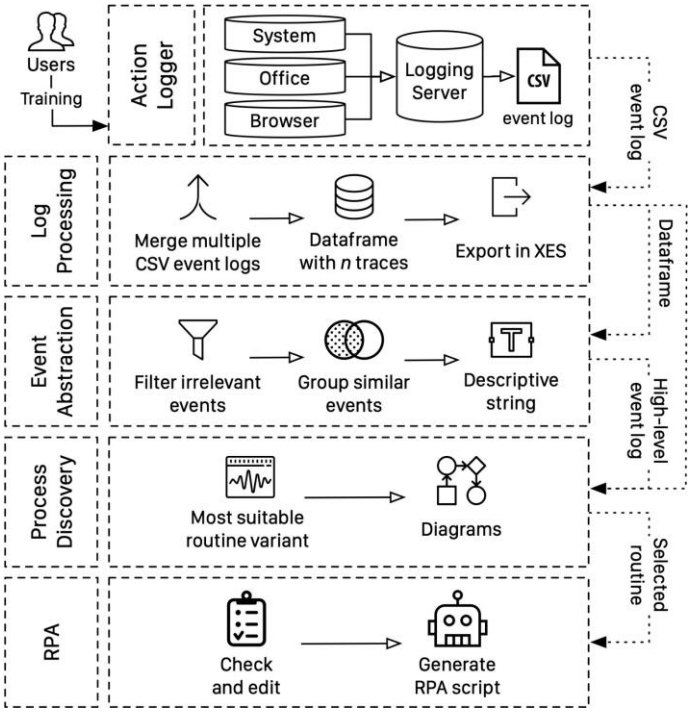
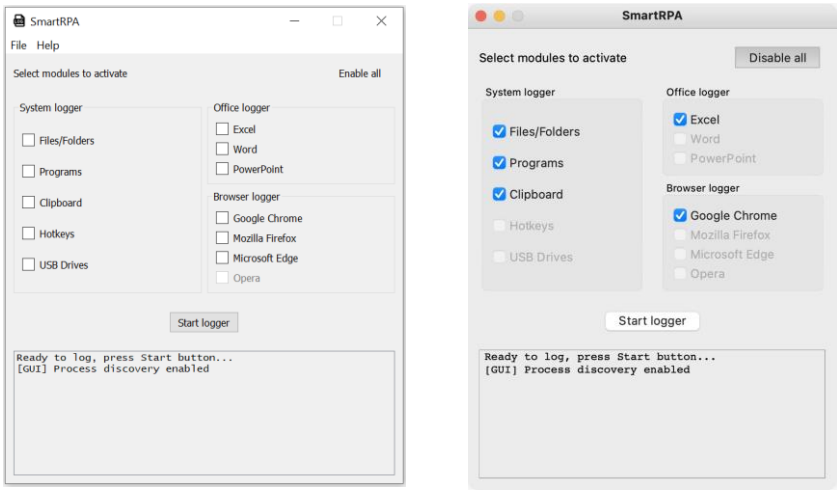


Figure 9: SmartRPA architecture

for any recorded combination of user inputs. An overview of the SmartRPA architecture is shown in Figure 9. The tool can be downloaded and tested at: https://github.com/bpm-diag/smartRPA

The first SW component of the architecture is an **Action Logger** that concretely implement the *Log Recording* step. The Action Logger provides a Graphical User Interface (GUI) that allows a user to select which SW applications s/he wants to record user actions on. All the applications that are not available in the host operating system of the user's computer are disabled by default. Then, the user can start the training session by clicking on the *"Start logger"* button, as shown in Figure 10. The Action Logger provides three categories of logging modules:

- *System Logger* : It detects those user actions not related to specific SW applications, i.e.: creation, renaming, movement and deletion of files/fold-



(a) Windows (b) MacOS

Figure 10: GUI of SmartRPA both on Windows and MacOS

ers; copy/paste of files/folders; opening/closing of applications; usage of double-click and hotkeys; insertion/remotion of USB drives.

- *Office Logger*: It detects the user actions performed within Microsoft Office applications, i.e.: Excel, Word and PowerPoint.
- *Browser Logger*: It detects the user actions performed on web browsers, i.e.: Google Chrome, Mozilla Firefox, Microsoft Edge and Opera.

Of course, multiple users can run the Action Logger on their computer system many times performing the same routine in different training sessions. When a training session is completed, i.e., when the routine of interest has been executed from the start to the end, the user can push the *"Stop logger"* button to stop the recording of user actions. The logging modules interact with a Logging Server implemented with the *Flask* framework,[5] which is in charge to store the user actions captured by the logging modules and organize them as *events* into

[5] https://palletsprojects.com/p/flask

several CSV[6] routine-based logs. Each CSV routine-based log contains exactly one (long) trace of user actions performed in a single training session by a single user. From a technical point of view, *(i)* system events are recorded using different Python modules, including *PythonCOM* (to access the Windows APIs and COM objects like the Microsoft Office suite), and *MacFSEvents* for MacOS; *(ii)* events generated by Microsoft Office applications are recorded using the Office JavaScript APIs; and *(iii)* browser events are recorded using dedicated JavaScript web extensions developed for each supported web browser.

In Figure 11, we show a snapshot of a CSV routine-based log recorded in one training session involving the execution of the routine presented in the running example.

	A	B	C	D	E	F	G	H	I	J	K	L
1	case:concept:name	case:creator	lifecycle:transition	time:timestamp	org:resource	category	application	concept:name	event_src_path	event_dest_path	clipboard_content	mouse_coord
2	1,10916E+15	SmartRPA by	complete	2020-11-09T15:5	marco	Browser	Chrome	startDownload				
3	1,10916E+15	SmartRPA by	complete	2020-11-09T15:5	marco	MicrosoftOffice	Microsoft Excel	openWorkbook	/Users/marco/desktop/travel authorization request procedure.xlsx			
4	1,10916E+15	SmartRPA by	complete	2020-11-09T15:5	marco	Browser	Chrome	reload				
5	1,10916E+15	SmartRPA by	complete	2020-11-09T15:5	marco	Clipboard	Clipboard	copy			Vanessa Costa	
6	1,10916E+15	SmartRPA by	complete	2020-11-09T15:5	marco	MicrosoftOffice	Microsoft Excel	editCell	/Users/marco/desktop/travel authorization request procedure.xlsx			
7	1,10916E+15	SmartRPA by	complete	2020-11-09T15:5	marco	Browser	Chrome	clickTextField				366,4
8	1,10916E+15	SmartRPA by	complete	2020-11-09T15:5	marco	Browser	Chrome	paste			Vanessa Costa	
9	1,10916E+15	SmartRPA by	complete	2020-11-09T15:5	marco	Browser	Chrome	changeField				
10	1,10916E+15	SmartRPA by	complete	2020-11-09T15:5	marco	Clipboard	Clipboard	copy			Full professor	
11	1,10916E+15	SmartRPA by	complete	2020-11-09T15:5	marco	MicrosoftOffice	Microsoft Excel	editCell	/Users/marco/desktop/travel authorization request procedure.xlsx			
12	1,10916E+15	SmartRPA by	complete	2020-11-09T15:5	marco	Browser	Chrome	clickTextField				346,498
13	1,10916E+15	SmartRPA by	complete	2020-11-09T15:5	marco	Browser	Chrome	paste			Full professor	
14	1,10916E+15	SmartRPA by	complete	2020-11-09T15:5	marco	Browser	Chrome	changeField				
15	1,10916E+15	SmartRPA by	complete	2020-11-09T15:5	marco	Clipboard	Clipboard	copy			ds_mail@uniroma1.it	
16	1,10916E+15	SmartRPA by	complete	2020-11-09T15:5	marco	MicrosoftOffice	Microsoft Excel	editCell	/Users/marco/desktop/travel authorization request procedure.xlsx			
17	1,10916E+15	SmartRPA by	complete	2020-11-09T15:5	marco	Browser	Chrome	mouseClick				256,579
18	1,10916E+15	SmartRPA by	complete	2020-11-09T15:5	marco	Browser	Chrome	paste			ds_mail@uniroma1.it	
19	1,10916E+15	SmartRPA by	complete	2020-11-09T15:5	marco	Browser	Chrome	changeField				
20	1,10916E+15	SmartRPA by	complete	2020-11-09T15:5	marco	Clipboard	Clipboard	copy			CNTCST19A58B176S	
21	1,10916E+15	SmartRPA by	complete	2020-11-09T15:5	marco	MicrosoftOffice	Microsoft Excel	editCell	/Users/marco/desktop/travel authorization request procedure.xlsx			
22	1,10916E+15	SmartRPA by	complete	2020-11-09T15:5	marco	Browser	Chrome	clickTextField				329,379
23	1,10916E+15	SmartRPA by	complete	2020-11-09T15:5	marco	Browser	Chrome	clickTextField				329,379
24	1,10916E+15	SmartRPA by	complete	2020-11-09T15:5	marco	Browser	Chrome	doubleClick				

Figure 11: Snapshot of the routine-based log captured during an execution of $R_{example}$

The second SW component of the architecture implements the **Log Processing** step. Specifically, after n training sessions, the Logging Server will deliver the n created CSV routine-based logs to the Log Processing component, which uses Algorithm 2 to import them into a single Pandas dataframe.[7] A dataframe is a two-dimensional size-mutable and heterogeneous tabular data

[6] CSV files are file formats that contain plain text values separated by commas. CSV files can be opened by any spreadsheet program, such as Microsoft Excel, Google Sheets, etc. CSV is only capable of storing a single sheet in a file, without any formatting and formulas.

[7] https://pandas.pydata.org/

Algorithm 2 Processing CSV routine-based logs

```
procedure processLogs(fileList )
    createDirectories()                          ▷ where files will be saved
    dfs ← list()                                 ▷ list of dataframes
    for any CSV log in fileList do
        df ← import CSV log into a DataFrame
        df ← rename columns to match XES standard
        df ← sort rows by timestamp
        df ← create case:concept:name column based on the first timestamp
        dfs.append(df )
    end for
    combinedDF ← combine all dataframes in dfs into a single one
    logXES ← export(combinedDF)                  ▷ exported as XES file
    return (combinedDF, logXES )
end procedure
```

structure with labeled axes (rows and columns), which is used as the main artifact to represent routine-based logs in SmartRPA. Of course, SmartRPA also produces an XES[8] (eXtensible Event Stream) version of the datastream that will contain exactly n traces, one for each recorded CSV routine-based log and can be inspected using the most popular process mining tools, such as *ProM*,[9] *Disco*[10] or *Apromore*.[11] The dataframe created by Algorithm 2 consists of low-level events with fine granularity associated one-by-one to a recorded user action (e.g., mouse clicks, file selections, etc.). Each row of the dataframe includes 45 columns with relevant data about the recorded event, i.e., its payload, such as: the timestamp, the application that generated the event, the resources involved, etc., cf. Figure 11.

[8] XES is the standard for the storage, interchange, and analysis of event logs [37]
[9] http://www.promtools.org/
[10] https://fluxicon.com/disco/
[11] https://apromore.org/

At this point, an **Event Abstraction** component is used to produce a high-level routine-based log from the low-level one, by performing the following steps:

1. *Filtering noise/irrelevant events.* The Action Logger records many low-level events in the dataframe-based routine-based log, such as the interaction with the browser windows (e.g., user actions “resize”, “open”, “close”), tabs (e.g., user actions “move”, “open”, “close”) and content (page zoom, installing extensions). From a workflow perspective, these events are not relevant for any RPA analyst that aims to understand the general behaviour of the routine. For this reason, they are filtered out by the high-level routine-based log under construction.

2. *Grouping similar events.* Within a dataframe-based routine-based log, different low-level events can refer to the same high-level concept. For example, in a web page, the Action Logger can capture 7 different types of clicks, based on the element that’s being clicked (“clickButton”, “clickTextField”, “doubleClick”, “clickTextField”, “mouseClick”, “clickCheckboxButton”, “clickRadioButton”). All these events just indicate that the user, during the training session, has clicked on an interactive element on the UI, thus the high-level workflow of the routine may just show the action “Click on button”, because from the RPA analyst perspective it is not relevant what kind of click was performed.

3. *Creating descriptive labels.* Any recorded event provides a low-level description of the nature of the user action performed. For example, if the user edits a cell in Excel, the Action Logger records one of these events: “editCellSheet”, “editCell”, or “editRange”. From the RPA analyst perspective, all such events refer to the same concept of “Editing a cell”. To this aim, to make the user action underlying an event more descriptive for the RPA analyst, further information (stored in the low-level dataframe-based routine-based log) can be added to its label, such as the cell and the sheet edited, the value inserted, etc. This allows us to create a (more)

descriptive label for any event in the high-level routine-based log, e.g., "*Edit cell B3 on Sheet 1 with value 'x'*".

Algorithm 3 Event Abstraction

```
procedure getHighLevelEvents(df: DataFrame)
    df ← filter irrelevant rows from df
    df ← group similar events in df
    for row in df do
        descriptiveRow ← create descriptive string for each event
    end for
    return a high-level dataframe-based routine-based log
end procedure
```

Concretely, the Event Abstraction component is realized enacting the above steps through Algorithm 3, and the outcome will be an high-level routine-based log to be used by the next component of the architecture.

At this point, the **Process Discovery** component of the architecture comes into play. Starting from the high-level routine-based log generated by the Event Abstraction component, it applies the heuristic miner algorithm (the decision to employ the heuristic miner has been driven by its ability to discover highly understandable flowcharts from a BPM analyst perspective [38]) implemented in PM4PY [39] to derive the high-level workflow describing the overall users' observed behavior as a Directly-Follows Graph (DFG). We show in Figure 12 a portion of the high-level workflow discovered from the high-level routine-based log associated to our running example.

Then, it applies Algorithm 1 (described in detail in Section 7) to automatically detect the different routine variants among all the routine traces stored in the low-level dataframe-based routine-based log, by evaluating the potential intermediate user inputs required to emulate the most suitable version of the routine on the UI.

Finally, there is the **Script Generation** component. Once the routine

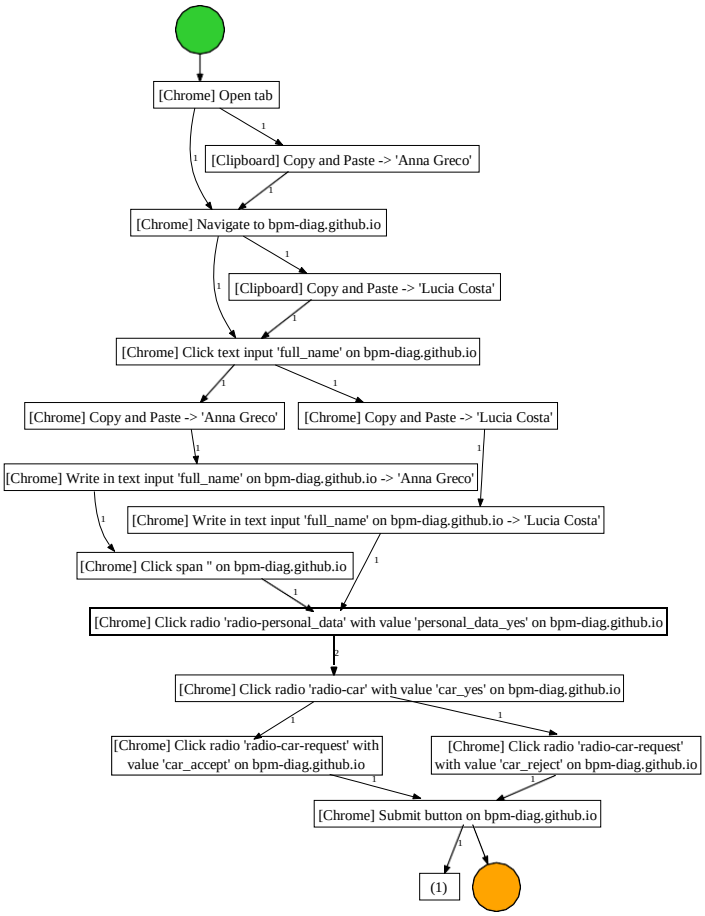


Figure 12: DFG describing a portion of the high-level workflow of $R_{example}$

variant to automatize is selected, before its enactment with a SW robot, it is possible for an RPA analyst to personalize the values stored in its events through a custom dialog window (cf. Figure 13).

The tool automatically detects the events that can be edited, such as typing something in a web page, renaming a file, pasting a text or editing an Excel cell, and dynamically builds the GUI to let the RPA analyst editing them. After confirmation, the low-level dataframe-based routine-based log is updated. Finally, the Python executable script based on the selected RPA routine and

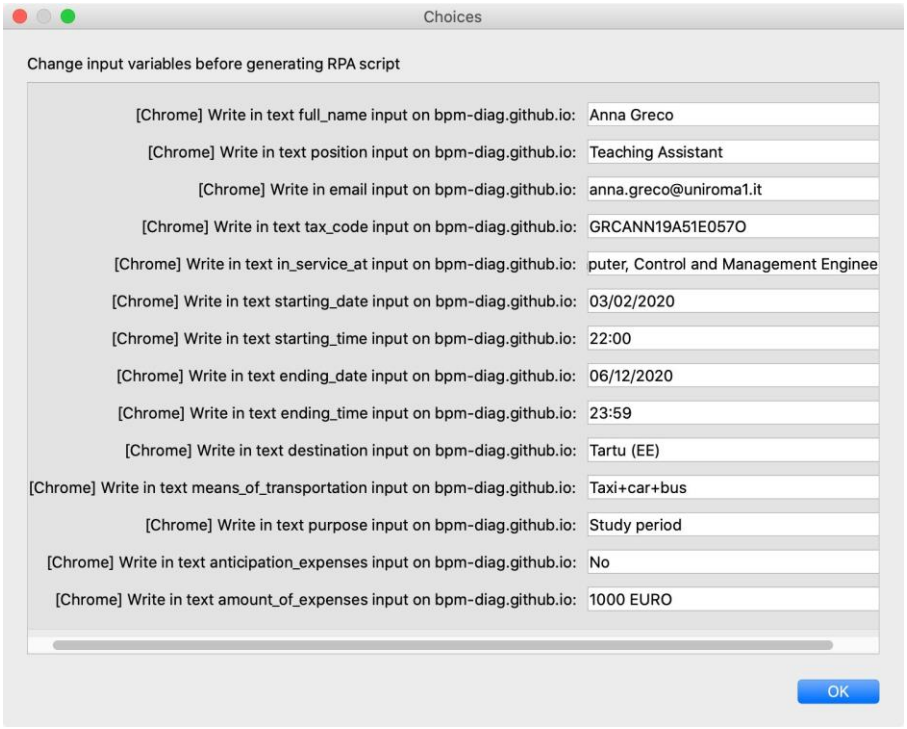


Figure 13: Custom dialog window to personalize editable fields of $R_{example}$

updated with the RPA analyst's edits, is generated by scanning the recorded low-level events in the dataframe-based log and converting them into executable pieces of SW code in Python, through Algorithm 4. To properly work the script generation algorithm relies on *Automagica*,[12] an Open Source framework for process automation, and *Selenium*,[13] a popular suite of tools for automating web browsers.

SmartRPA is also able to generate executable RPA scripts compatible with *UiPath*, a tool that allows to design automation processes in a visual manner. Once the routine variant to automate along with the RPA analyst's edits has been generated, the UiPath script is accordingly created. UiPath files are

[12] https://github.com/automagica/automagica
[13] https://www.selenium.dev/

Algorithm 4 Python Script Generation

```
1: procedure generatePythonSWRobot(df: DataFrame)
2:     for row in df do
3:         pythonEvent ← generatePythonEvent(row)
4:         write pythonEvent to Python file
5:     end for
6: end procedure
```

written in XAML (Extensible Application Markup Language), a declarative language based on XML. A sample XAML file is shown in Figure 14. It is composed by a *Main Sequence* containing in turn multiple sequences, based on the category of the user actions. For example, all the user actions related to the browser should be wrapped by a *Browser Activities* sequence, because they all share the same browser. Likewise, all the user actions from Excel should go into the *Excel Activities* sequence, because they all refer to an Excel spreadsheet. The same thing applies for *System* and *Microsoft Office* user actions. Every sequence contains a series of activities. An *activity* is a block of XML code

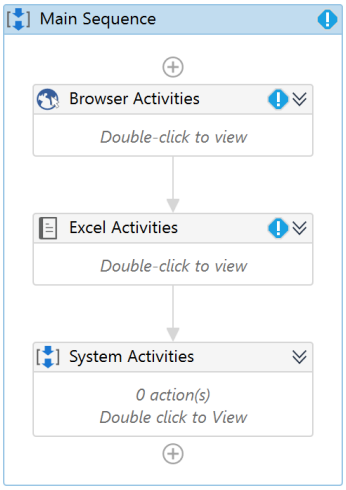


Figure 14: UiPath sequence

with a list of parameters. In order to generate the SW robot, XML activities are generated from each event in the low-level dataframe-based routine-based log using lxml[14] Python library. Activities are created with Python methods which take parameters as input and return XML nodes, as shown in Algorithm 5. The algorithm describes the process to generate a UiPath SW robot from a given low-level dataframe-based routine-based log. The main XMLsequence

Algorithm 5 UiPath Script Generation

```
1: procedure generateUiPathSWRobot(df: DataFrame)
2: mainSequence ← create main XML sequence
3:     activities ← dict()                          ▷ dictionary to store XML activities
4:     lastIndex ← False                            ▷ True in the last loop iteration
5:     categoryChange ← False                       ▷ True when there is a change in category 6:
for row in df do
7:         XMLNode ← generateXMLNode(row)
8:         activities[currentCategory].append(XMLNode)
9:         if categoryChange or lastIndex then
10:            mainSequence.append(createSequence(activities['Browser']))
11:            mainSequence.append(createSequence(activities['MicrosoftOffice']))
12:            mainSequence.append(createSequence(activities['OperatingSystem']))
13:            activities.clear()                   ▷ empty dictionary
14:        end if
15: end for
16:    write mainSequence to XAML file              ▷ UiPath Project
17: end procedure
```

that will contain all other sequences is created, along with a dictionary to store activities based on their category (*browser, office, system*). For each event in the low-level dataframe-based routine-based log, a corresponding XML node is generated and appended to the dictionary based on its category. When there

[14] https://lxml.de/

is a change in category and in the last loop iteration, user actions are wrapped in a sequence specific for that category and added to the main sequence. The *activities* dictionary is cleared before restarting the loop, to prevent duplicate activities in the main sequence. Finally, the generated XML sequence is written as a XAML file that can be opened and run in UiPath.

A screencast with installation instructions and showing the working of SmartRPA is available in the github repository of the tool at: https://github.com/bpm-diag/smartRPA/.

9. Evaluation

In this section, we present the results of a multi-step evaluation performed on SmartRPA to investigate the extent to which our approach satisfies four relevant non-functional requirements, namely *robustness*, *feasibility*, *effectiveness* and *usability*. The target is to understand if SmartRPA can potentially complement the traditional model-based solutions provided by commercial RPA tools.

9.1. Assessing the Robustness and Feasibility of the Algorithm for the Automated Detection of Variation Points

To investigate the *robustness* and *feasibility* of our approach to the reactive synthesis of SW robots from UI logs, we performed several synthetic experiments employing UI logs of increasing complexity. Specifically, we generated 240 different UI logs (containing in total 150.000 different routine traces), in a way that each UI log was characterized through a unique configuration obtained by varying the following input settings:

- *log_size*: number of traces in the UI log (250/500/750/1000);
- *trace_size*: number of events in each routine trace (25/50/75/100);
- *events size*: number of possible different events to be considered for the creation of a trace (40/80/120);

- *variation points*: number of different variation points included in the UI log (1/2/3/4/5).

Note that the amount of possible decisions to be taken in a variation point was generated randomly, ranging from 2 to 10 possible outgoing decisions. Following our definition of variation points explained in Section 3.1, each UI log was generated creating log traces having a similar structure in terms of recorded events, except for the presence of the variation points. Repeated events and concurrency are allowed inside a UI log. However, since they have been randomly introduced in UI logs, we can not provide solid findings related to their impact on the identification of the variation points. The synthetic UI logs generated for the test are available for testing and experiments repeatability at: https://doi.org/10.5281/zenodo.6518291.

The target was to investigate if the amount and anatomy of variation points discovered by SmartRPA is the same that was synthetically introduced in the sample routine executions recorded in the UI logs (i.e., *robustness*), and to measure the performance of the entire approach to generate a SW robot by solely using the UI logs (i.e., *feasibility*).

Concerning the robustness of the approach, for all the 240 tested logs the approach was able to always discover the correct variation points to be considered for the synthesis of SW robots. It is worth noticing that this result is justified by the fact that we employed a fixed (yet large) alphabet of user actions for the generation of the sample UI logs, in line with the assumption that a routine reflects highly predictable and repetitive work with low flexibility requirements (and, consequently, with a low and predictable number of variants) [21]. On the other hand, in case of more flexible procedures, our algorithm would detect a new variation point any time there are distinct user actions (cf. Section 7) recorded at the same point of different executions of the same routine. This would lead to a consistent growth of the amount of identified variation points, which is not wrong in principle, but that could not be suitable to concisely represent the behaviour of a routine. Therefore, we can state that our algorithm

Event size: 40	Time				
Trace size	1	2	3	4	5
25	0.453	0.452	0.53	0.409	0.423
50	0.417	0.433	0.417	0.425	0.419
75	0.439	0.511	0.424	0.43	0.431
100	0.454	0.416	0.421	0.424	0.431
Event size: 80	Time				
Trace size	1	2	3	4	5
25	0.422	0.428	0.43	0.413	0.412
50	0.427	0.425	0.444	0.417	0.428
75	0.42	0.428	0.553	0.422	0.437
100	0.442	0.434	0.428	0.438	0.432
Event size: 120	Time				
Trace size	1	2	3	4	5
25	0.413	0.507	0.421	0.416	0.421
50	0.421	0.412	0.417	0.42	0.421
75	0.425	0.433	0.438	0.451	0.429
100	0.437	0.433	0.428	0.532	0.523

Table 2: Experimental results showing the *feasibility* of SmartRPA to the reactive generation of SW robots (only logs with 1000 traces are shown here). The time (in *seconds*) is the average per trace.

for the detection of variation points is *robust* if the UI logs have the features outlined in the experiment settings. In absence of further experiments, we can not state anything about the robustness of the algorithm when our working assumptions are contradicted, i.e., when more flexible (i.e., non repetitive) procedures are executed. Note that the literature proposes dedicated approaches to detect the decision points in case of less flexible processes to be analyzed, e.g., see [40], even if the granularity of the process activities is less fine than the one of the user actions involved in a routine execution.

Concerning the feasibility, it was measured in terms of the computation time required to generate a SW robot starting from UI logs of growing complexity.

The results, which are summarized in Table 2, indicate that the total computation time grows with the number of traces in the UI log,[15] ranging from ~ 100ms for UI logs with 250 traces up to ~ 500ms for event logs with 1000 traces. This result was expected, since more traces in a UI log means more executions to analyze and interpret. On the other hand, if we consider a fixed log size, it seems that the performance of the approach scales very well in case of an increasing number of variation points to be discovered and log traces/alphabet of events of growing size. Sometimes, it has been also observed that SmartRPA gets faster by adding events in a trace, which suggests that the performance of the approach does not suffer the presence of a larger alphabet of events.

9.2. Evaluating the Effectiveness of SmartRPA

In order to address **RQ3**, we enacted a controlled experiment involving real users exploiting the use case of our running example to investigate the effectiveness of the SmartRPA approach when compared to UiPath, which is one of the major vendors in the RPA market according to [10], and realizes the "traditional" model-based approach for the generation of SW robots.

To this end, we conducted a user study based on the running example presented in Section 4, by asking to 20 different administration employees to fill the Google Form using the data from the Excel spreadsheet containing the information to apply for a travel request. All the user actions were enacted on distinct computer systems having different features and operating system. During the execution of their routine, the employees were coupled with a first group of 20 (out of a sample of 40) Master students of the course of Process Management and Mining (PMM) held at Sapienza University of Rome (one student per employee), which were requested to observe the execution steps of $R_{example}$. We denote with p_1 this first group of users. In parallel, a second group of 20 users

[15] For the sake of space, the table includes only the results related to UI logs containing 1000 traces. The complete list of results can be analyzed at: https://doi.org/10.5281/zenodo.6518291

were remotely connected to the computer systems of the employees, with the target to record the user actions performed on the UI of such systems exploiting the Action Logger component of SmartRPA, thus generating at the end 20 different UI logs. We denote with p_2 this second group of users. It is worth noticing that all the PMM students involved in the user study can be considered as *expert users* in business process modeling and automation.

At this point, we requested to any of the 20 expert users in p_1 to employ UiPath to model a flowchart diagram associated to $R_{example}$ and generate the associated SW robot using the functionalities of the UiPath framework. On the other hand, we asked to any of the 20 expert users in p_2 to exploit the UI logs storing the executions of $R_{example}$ as inputs to use SmartRPA for the generation of the associated SW robot.

To assess the effectiveness of SmartRPA to synthesize SW robots from UI logs, we investigated the following experimental hypothesis $\mathbf{H}_1$: *Employing the SmartRPA approach, thus neglecting the manual specification stage of the routine behavior, is more effective than employing traditional approaches that require to manually specify and implement the behaviour of SW robots by means of flowchart models*. To this aim, we have first built the null hypothesis $\mathbf{H}_0$: *Employing the SmartRPA approach does not provide any advantage in terms of effectiveness if compared with traditional modeling-driven RPA approaches*. Then, to support or reject $\mathbf{H}_0$, a *between-subject approach* was used, i.e., each user in p_1 (p_2, respectively) was assigned to a different experimental condition, related to the exclusive use of UiPath (c_1) or SmartRPA (c_2) to perform the required steps for the generation of the SW robot for $R_{example}$. Any user in p_1 (p_2, respectively) was preliminarily instructed about the functionalities of UiPath (SmartRPA, respectively) through a short training session. Notice that we selected users that were completely unaware about the use of both UiPath and SmartRPA before the starting of the experiment.

We evaluated the validity of $\mathbf{H}_0$ by asking any expert user that completed the user study the following three questions:

- Q_1: The development life-cycle of a SW robot (from the definition of the routine behavior to the generation and execution of the associated SW robot) is a time-consuming task. Do you agree?
- Q_2: The extraction of the routine's knowledge required for the development and execution of a SW robot is a complex task. Do you agree?
- Q_3: Once a SW robot has been generated, the monitoring of its execution and the inspection of its behaviour is a complex task. Do you agree?

Questions are rated with a 7-point average numerical scale structured as follows: 1 ("Strongly Disagree"), 2 ("Disagree"), 3 ("Somewhat Disagree"), 4 ("Neither Agree nor Disagree"), 5 ("Somewhat Agree"), 6 ("Agree"), 7 ("Strongly Agree"). We kept the same difference (numerical 1) between subsequent points of the scale, as suggested by [41]. The choice to employ a 7-point scale (rather than a 5-point scale) is supported by the findings of Sauro [42], which states that in case of a questionnaire consisting of few questions "*having seven points tends to be a good balance between having enough points of discrimination without having to maintain too many response options*".

To evaluate the answers associated to Q_1, Q_2 and Q_3 we performed a comparison of the rates obtained from the questionnaire, respectively in the cases of c_1 and c_2. Specifically, for each question, we employed a *2-Sample t-test* with a 95% confidence level to determine whether the means between the two distinct populations (i.e., independent groups p_1 and p_2) involved in c_1 and c_2 differ. Before running the 2-Sample t-test, we first exploited the Kolmogorov Smirnov Statistic (KS Test) to establish the normality of the distribution of the collected data [43], and then we checked that the variances and standard deviations in both groups were approximately equal [42].

Finally, we measured the level of statistical significance analyzing the resulting *p-value*, choosing 0.05 as the threshold value. The results of the analysis are summarized in Figure 15. It appears evident that the null hypothesis H_0 is statistically supported by the results obtained for Q_3, while it is rejected for

Q1		Q2		Q3	
SmartRPA	UiPath	SmartRPA	UiPath	SmartRPA	UiPath
2	3	2	3	3	1
2	3	2	3	3	2
3	3	3	3	3	2
3	3	3	3	3	2
3	4	3	3	3	3
3	4	3	4	3	3
3	4	3	4	4	3
3	4	3	4	4	3
4	4	3	4	4	3
4	4	4	4	4	3
4	4	4	4	4	3
4	4	4	4	4	4
4	4	4	4	4	4
4	4	4	4	4	4
4	5	4	5	4	4
4	5	4	5	4	4
4	5	4	5	5	4
4	5	4	6	5	4
5	6	5	6	6	4
5	6	5	6	6	4
p-value	0,0333436	**p-value**	0,0317368	**p-value**	0,0081553

Figure 15: Effectiveness of SmartRPA: *p*-values associated to each question.

Q_1 and Q_2. Concerning Q_3, there is a strong evidence that a traditional model-based approach based on designing routines by means of flowchart diagrams (like UiPath) is more effective to monitor the behaviour of the (running) SW robots associated to the routines and to inspect the related RPA scripts. On the other hand, to skip completely the modeling task by employing an approach based only on UI logs enables a faster generation of SW robots (cf. Q_1) requiring solely the knowledge stored in the UI logs (cf. Q_2). In summary, we can conclude that log-based approaches like SmartRPA increase the degree of automation of the design-time steps required to generate SW robots, reducing the intervention of human experts in this phase. Therefore, H_1 can be considered as validated for Q_1 and Q_2 but rejected for Q_3, where model-based approaches appear to be more effective to monitor the working of the running SW robots.

9.3. Quantifying the Usability of the UI of SmartRPA

Last but not least, we investigated the degree of *usability* of the UI developed for SmartRPA. Specifically, we administered the SUS (Software Usability Scale) questionnaire (which is one of the most widely used methodology to measure

the users' perception of the usability of a tool [44]) to the 20 expert users that were involved in the experimental condition *c2*, i.e., that used SmartRPA. The questionnaire consists of 10 statements evaluated with a 5-point numerical scale that ranges from 1 ("strongly disagree") to 5 ("strongly agree"). At the end of the questionnaire, an overall score is assigned to the questionnaire. The score can be compared with several benchmarks presented in the research literature to determine the degree of usability of the tool being evaluated. In our test, we made use of the benchmark presented in [42], which associates to each range of the SUS score a percentile ranking varying from 0 to 100, indicating how well it compares to other 5,000 SUS observations performed in the literature.

The collection of the ranks associated to any statement of the SUS is reported in Figure 16, calculated following the steps discussed in [42]. Since the average SUS score obtained by the tool was 79.3, according to the selected benchmark [42], the usability of the tool corresponds to a rank of A-, which indicates a degree of usability among very good and excellent.

Participant	q1	q2	q3	q4	q5	q6	q7	q8	q9	q10	SUS Score	Average
p1	4	2	4	2	4	2	4	2	4	2	75,0	79,3
p2	4	1	4	2	4	2	3	2	4	2	75,0	
p3	3	1	4	1	4	2	5	1	1	1	77,5	
p4	4	1	1	1	4	2	4	2	4	1	75,0	
p5	3	1	5	1	4	2	4	2	4	1	82,5	
p6	4	2	5	2	4	1	4	2	5	1	85,0	
p7	1	1	5	1	4	2	5	1	5	1	85,0	
p8	3	2	4	2	4	2	4	2	4	2	72,5	
p9	4	2	5	1	4	2	4	1	4	1	85,0	
p10	4	2	4	2	4	2	4	2	5	2	77,5	
p11	4	2	4	2	4	2	4	2	4	2	75,0	
p12	4	2	4	2	4	2	4	2	4	2	75,0	
p13	3	2	4	2	4	2	4	2	4	2	72,5	
p14	4	1	4	2	4	2	4	2	4	2	77,5	
p15	4	2	2	2	3	1	4	2	4	1	72,5	
p16	4	1	5	2	4	2	5	2	4	2	82,5	
p17	4	2	4	1	4	3	4	1	4	2	77,5	
p18	4	2	4	1	3	3	4	1	4	1	77,5	
p19	4	1	5	1	5	1	5	1	4	2	92,5	
p20	5	2	5	2	4	1	5	1	5	1	92,5	

Figure 16: Computation of the SUS overall score

9.4. Threats to Validity

A series of common issues may influence the results of our evaluation, such as the (random) selection of the sample of users who performed the experiments (even if from a well-defined population, which mitigates the issue), the selection

of the statistical tests to evaluate the collected data, etc.

Notably, while the controlled experiments employed to measure the effectiveness of SmartRPA appears to have an high internal validity due to the control of the experimental conditions exercised throughout the experiment, on the other hand, this control can cause the experiment to have a questionable external validity. This is due to the complexity to replicate the experimental conditions in real-world settings that have many extraneous variables at play, making the findings less generalizable. However, we observe that we do not claim that our results are representative of all RPA literature, or to be generalizable to other fields or contexts.

Concerning the experiments' findings, we claim that their validity is bound to the experiments settings. For example, in the case of the experiment to measure the effectiveness of SmartRPA, using a 2-Sample t-test with a 95% confidence level enables us to state that we are 95% confident that the null hypothesis H_0 is partially rejected. However, performing a further experiment that includes more users and the application of a second confidence level (e.g., set to 99%) could support more substantial evidence of the results.

10. Conclusion

While RPA is currently used for automating routines and high-volume tasks requiring a manual intervention of expert users, the aim of SmartRPA is to automatically develop SW robots directly from the users' observed behavior. SmartRPA offers an innovative contribution to RPA technology with the goal of mitigating some of its core downsides.

In this paper, we leverage a design science research method [19] to build the SmartRPA approach, which is able to interpret the UI logs keeping track of many routine executions, and to automatically synthesize SW robots that emulate the most suitable routine variant for any specific intermediate user input that is required during the routine execution.

Notably, using SmartRPA, all the routine executions recorded by the tool

can be automated, an high-level flowchart diagram is presented to expert users for potential diagnosis operations, and the executable RPA scripts to drive the working of a SW robot are generated by solely interpreting the routine executions stored in the routine-based log, selecting step-by-step the most suitable routine variant.

From a technical perspective, the script generation algorithm takes into account only the platform where the SW robot is going to be run, regardless of the operating system used to capture the log. For example, if the selected routine variant was recorded on MacOS, but the tool is being executed on Windows, the RPA script will be generated taking into account this aspect, e.g., by converting the information about the system paths. This guarantees cross-platform compatibility across UI logs recorded on different platforms, as suggested by the guidelines principles of RPM. Last but not least, SmartRPA creates executable RPA scripts also for UiPath. These scripts can then be executed via the interface of UiPath. In addition, the tool allows us to personalize some input fields of the selected routine variant before executing the related RPA scripts (either on Windows/MacOS systems or within UiPath Studio), thus supporting those steps that require intermediate manual user inputs. As a consequence, this makes the working of SW robots flexible and adaptable to several real-world situations. To sum up, we consider SmartRPA as an important first step towards the intelligent fully automated generation of SW robots.

The main weakness of the approach is correlated with the quality of information recorded in real-world UI logs. Since a UI log is fine-grained, routines executed with many different strategies may potentially affect the robustness of our approach to the detection of variation points. For this reason, as a future work, we are going to perform a robust evaluation of the algorithm on further real-world case studies including heterogeneous UI logs obtained from different application domains.

Moreover, in SmartRPA we have currently neglected the *segmentation* issue [18, 45]. Segmentation is the challenge to automatically understand which user actions contribute to which routines inside a UI log, which is trivially already

solved in SmartRPA because UI logs are generated through controlled training sessions. Nonetheless, as a future work, we aim at including a segmentation component in the SmartRPA architecture to enable the analysis of unsegmented UI logs obtained in more inclusive training sessions that are not focused just on single routines. While partial solutions to the segmentation issue have been recently explored in [46, 47], we also plan to investigate how to integrate existing Artificial Intelligence based solutions in BPM that address similar research challenges, cf. [48, 49].

This paper extends previous work in [17] in several directions and includes many new elements that were previously neglected:

- A revised introduction that makes immediately clear for the reader the research problem to be tackled, its significance in the RPA field, and the proposed contribution to solve the problem, driven by three main research questions (cf. Section 1).
- A new section that describes the phases of our design-science based research methodology (cf. Section 2).
- A new section that makes the background and the relevant preliminary concepts explicit (cf. Section 3);
- A revised related work section where we discuss the relevant state-of-the-art approaches that are able to mitigate the research challenges, and we derive a set of technical requirements to realize our SmartRPA approach (cf. Section 5).
- A new section (cf. Section 7) presenting an algorithm to the automated identification of the variation points of a routine, to enable the selection of the most suitable routine variants to be implemented with a SW robot (thus, not just the most frequent one, removing the most evident weakness of our previous work);
- An improved description of the various architectural components of the tool (and in particular of the script generation algorithm) to provide the

reader with a complete understanding of the proposed approach, and a new contribution describing how the generated scripts can be automatically encoded in a format readable by the commercial RPA tool UiPath (cf. Section 8);

- A robust evaluation section to investigate the extent to which the proposed approach satisfies four relevant non-functional requirements, namely robustness, feasibility, effectiveness and usability, employing both synthetic and real-world datasets (cf. Section 9);
- All other sections of the previous work have been edited and refined to present the material more thoroughly.

Acknowledgments. This work has been supported by the "Dipartimento di Eccellenza" grant, the H2020 project DataCloud and the Sapienza grant BPbots.

References

[1] W. M. P. van der Aalst, M. Bichler, A. Heinzl, Robotic Process Automation, Bus. Inf. Syst. Eng. 60 (4) (2018) 269–272. doi:10.1007/s12599-018-0542-4.

[2] E. Penttinen, H. Kasslin, A. Asatiani, How to Choose between Robotic Process Automation and Back-end System Automation?, in: European Conference on Information Systems (ECIS), 2018.
URL https://aisel.aisnet.org/ecis2018_rp/66

[3] M. Lacity, L. P. Willcocks, A. Craig, RPA at Telefonica O2, The London School of Economics and Political Science, 2015.

[4] S. Aguirre, A. Rodriguez, Automation of a Business Process Using Robotic Process Automation (RPA): A Case Study, in: Applied Computer Sciences in Engineering, Springer, 2017, pp. 65–71. doi:10.1007/978-3-319-66963-2_7.

[5] S. Anagnoste, Setting up a Robotic Process Automation Center of Excellence, Management Dynamics in the Knowledge Economy 6 (2) (2018) 307–332.

[6] J. Kokina, S. Blanchette, Early Evidence of Digital Labor in Accounting: Innovation with Robotic Process Automation, International Journal of Accounting Information Systems 35 (2019). doi:10.1016/j.accinf.2019.100431.

[7] M. Schmitz, C. Dietze, C. Czarnecki, Enabling Digital Transformation through Robotic Process Automation at Deutsche Telekom, in: Digitalization Cases, Springer, 2019, pp. 15–33. doi:10.1016/j.accinf.2019.100431.

[8] M. Smeets, R. Erhard, T. Kaußler, et al., Robotic Process Automation (RPA) in der Finanzwirtschaft, Springer Books, 2019. doi:10.1007/ 978-3-658-26564-9.

[9] C. Langmann, D. Turi, et al., Robotic Process Automation (RPA)-Digitalisierung und Automatisierung von Prozessen, Springer Books, 2020. doi:10.1007/978-3-658-28299-8.

[10] C. Dilmegani, 55 RPA Software Tools & Vendors of 2021. AI-Multiple, https://blog.aimultiple.com/rpa-tools/, Accessed: 19-07-2021 (2021).

[11] S. Agostinelli, A. Marrella, M. Mecella, Research Challenges for Intelligent Robotic Process Automation, in: Business Process Management Workshops, BPM'19, Springer, 2019, pp. 12–18. doi:10.1007/978-3-030-37453-2_2.

[12] S. Agostinelli, A. Marrella, M. Mecella, Towards Intelligent Robotic Process Automation for BPMers, in: AAAI-20 Workshop on Intelligent Process Automation, Vol. abs/2001.00804, 2020. arXiv:2001.00804.

[13] T. Chakraborti, V. Isahagian, R. Khalaf, Y. Khazaeni, V. Muthusamy, Y. Rizk, M. Unuvar, From Robotic Process Automation to Intelligent Process Automation: Emerging Trends, in: Business Process Management: Blockchain and Robotic Process Automation Forum - BPM '20, Springer, 2020, pp. 215–228. doi:10.1007/978-3-030-58779-6_15.

[14] A. Jimenez-Ramirez, H. A. Reijers, I. Barba, C. Del Valle, A Method to Improve the Early Stages of the Robotic Process Automation Lifecycle, in: 31st Int. Conf. on Advanced Information Systems Engineering, CAiSE'19, 2019, pp. 446–461. doi:10.1007/978-3-030-21290-2_28.

[15] L.-V. Herm, C. Janiesch, A. Helm, F. Imgrund, K. Fuchs, A. Hofmann, A. Winkelmann, A Consolidated Framework for Implementing Robotic

Process Automation Projects, in: 18th Int. Conf. on Business Process Management, BPM'20, Springer, 2020, pp. 471–488. doi:10.1007/978-3-030-58666-9_27.

[16] R. Ravn, P. Halberg, J. Gustafsson, J. Groes, Get ready for robots: why planning makes the difference between success and disappointment, https://eyfinancialservicesthoughtgallery.ie/wp-content/uploads/2016/11/ey-get-ready-for-robots.pdf, Accessed: 19-07-2021 (2016).

[17] S. Agostinelli, M. Lupia, A. Marrella, M. Mecella, Automated Generation of Executable RPA Scripts from User Interface Logs, in: Business Process Management: Blockchain and Robotic Process Automation Forum - BPM 20', Springer, 2020, pp. 116–131. doi:10.1007/978-3-030-58779-6_8.

[18] V. Leno, A. Polyvyanyy, M. Dumas, M. La Rosa, F. M. Maggi, Robotic Process Mining: Vision and Challenges, Bus. Inf. Syst. Eng. March '20 (2020) 1–14. doi:10.1007/s12599-020-00641-4.

[19] P. Johannesson, E. Perjons, An Introduction to Design Science, Springer, 2014.

[20] J. Bisbal, D. Lawless, B. Wu, J. Grimson, Legacy Information Systems: Issues and Directions, IEEE Software 16 (5) (1999) 103–111. doi:10.1109/52.795108.

[21] C. Di Ciccio, A. Marrella, A. Russo, Knowledge-Intensive Processes: Characteristics, Requirements and Analysis of Contemporary Approaches, Journal on Data Semantics 4 (1) (2015) 29–57. doi:10.1007/ s13740-014-0038-4.

[22] C. Tornbohm, R. Dunie, Market Guide for Robotic Process Automation Software, Gartner Dec '17 (2017) (2017).

[23] W. M. P. van der Aalst, Process Mining: Data Science in Action, 2nd Edition, Springer, Heidelberg, 2016. doi:10.1007/978-3-662-49851-4.

[24] V. Leno, A. Polyvyanyy, M. L. Rosa, M. Dumas, F. M. Maggi, Action Logger: Enabling Process Mining for Robotic Process Automation, in: Proceedings of the Dissertation Award, Doctoral Consortium, and Demonstration Track at 17th Int. Conf. on Business Process Management, (BPM'19), 2019, pp. 124–128.
URL http://ceur-ws.org/Vol-2420/paperDT2.pdf

[25] A. Bosco, A. Augusto, M. Dumas, M. L. Rosa, G. Fortino, Discovering Automatable Routines from User Interaction Logs, in: Business Process Management Forum - BPM Forum 2019, 2019, pp. 144–162. doi:10.1007/978-3-030-26643-1_9.

[26] C. Linn, P. Zimmermann, D. Werth, Activity Mining - A new level of detail in mining business processes, in: Workshops der INFORMATIK 2018 - Architekturen, Prozesse, Sicherheit und Nachhaltigkeit, 26.-27, 2018, pp. 245–258.
URL https://dl.gi.de/20.500.12116/17225

[27] J. Gao, S. J. van Zelst, X. Lu, W. M. P. van der Aalst, Automated Robotic Process Automation: A Self-Learning Approach, in: On the Move to Meaningful Internet Systems: OTM 2019 Conf., Springer, 2019, pp. 95–112.
URL https://doi.org/10.1007/978-3-030-33246-4_6

[28] V. Le, S. Gulwani, FlashExtract: A Framework for Data Extraction by Examples, in: ACM SIGPLAN PLDI '14, 2014, pp. 542–553.
URL https://doi.org/10.1145/2594291.2594333

[29] A. Miltner, S. Gulwani, V. Le, A. Leung, A. Radhakrishna, G. Soares, A. Tiwari, A. Udupa, On the Fly Synthesis of Edit Suggestions, In: ACM Program. Lang. 3 (OOPSLA) (2019) 143:1–143:29. doi:10.1145/3360569.

[30] V. Leno, S. Deviatykh, A. Polyvyanyy, M. L. Rosa, M. Dumas, F. M. Maggi, Robidium: Automated Synthesis of Robotic Process Automation Scripts from UI Logs, in: Demonstration & Resources Track at 18th International

Conference on Business Process Management (BPM 2020), 2020, pp. 102–106.
URL http://ceur-ws.org/Vol-2673/paperDR08.pdf

[31] N. Ito, Y. Suzuki, A. Aizawa, From Natural Language Instructions to Complex Processes: Issues in Chaining Trigger Action Rules, arXiv CoRR abs/2001.02462 (2020). arXiv:2001.02462.
URL https://arxiv.org/abs/2001.02462

[32] H. Leopold, H. van der Aa, H. A. Reijers, Identifying Candidate Tasks for Robotic Process Automation in Textual Process Descriptions, in: Int. Conf. on Bus. Proc. Mod., Dev. and Supp. (BPMDS'18), Springer, 2018, pp. 67–81. doi:10.1007/978-3-319-91704-7_5.

[33] X. Han, L. Hu, Y. Dang, S. Agarwal, L. Mei, S. Li, X. Zhou, Automatic Business Process Structure Discovery using Ordered Neurons LSTM: A Preliminary Study, arXiv CoRR abs/2001.01243 (2020). arXiv:2001.01243.
URL https://arxiv.org/abs/2001.01243

[34] A. Ayub, A. R. Wagner, Teach Me What You Want to Play: Learning Variants of Connect Four through Human-Robot Interaction (2020). arXiv:2001.01004.
URL https://arxiv.org/abs/2001.01004

[35] P. Jenkins, H. Wei, J. S. Jenkins, Z. Li, A Probabilistic Simulator of Spatial Demand for Product Allocation (2020). arXiv:2001.03210.
URL https://arxiv.org/abs/2001.03210

[36] A. Dix, J. E. Finlay, G. D. Abowd, R. Beale, Human-Computer Interaction, 3rd Edition, Prentice-Hall, Inc., USA, 2003.

[37] IEEE Digital Library, Standard for eXtensible Event Stream (XES) for Achieving Interoperability in Event Logs and Event Streams, IEEE Std 1849-2016 (2016). doi:10.1109/IEEESTD.2016.7740858.

[38] S. Agostinelli, F. M. Maggi, A. Marrella, F. Milani, A User Evaluation of Process Discovery Algorithms in a Software Engineering Company, in: 2019 IEEE 23rd International Enterprise Distributed Object Computing Conference (EDOC), 2019, pp. 142–150. doi:10.1109/EDOC.2019.00026.

[39] A. Berti, S. J. van Zelst, W. van der Aalst, Process Mining for Python (PM4Py): Bridging the Gap Between Process- and Data Science (2019). arXiv:1905.06169.

[40] M. de Leoni, M. Dumas, L. García-Bañuelos, Discovering Branching Conditions from Business Process Execution Logs, in: 16th International Conference on Fundamental Approaches to Software Engineering (FASE'13), 2013, pp. 114–129. doi:10.1007/978-3-642-37057-1_9.

[41] A. Dix, Statistics for HCI: Making Sense of Quantitative Data, Synthesis Lectures on Human-Centered Informatics 13 (2) (2020) 1–181.

[42] J. Sauro, J. R. Lewis, Quantifying the User Experience: Practical Statistics for User Research, Morgan Kaufmann, 2016. doi:10.1145/2413038.2413056.

[43] I. M. Chakravarti, R. G. Laha, J. Roy, Handbook of methods of applied statistics, Wiley Series in Probability and Mathematical Statistics (USA) (1967).

[44] J. Brooke, SUS: a Retrospective, Journal of Usability Studies 8 (2) (2013) 29–40.

[45] S. Agostinelli, A. Marrella, M. Mecella, Exploring the Challenge of Automated Segmentation in Robotic Process Automation, in: 15th International Conference on Research Challenges in Information Science (RCIS 2021), Springer, 2021, pp. 38–54. doi:10.1007/978-3-030-75018-3_3.

[46] S. Agostinelli, A. Marrella, M. Mecella, Automated segmentation of user interface logs, De Gruyter Oldenbourg, 2021, pp. 201–222. doi:doi:10.1515/9783110676693-011.

[47] S. Agostinelli, F. Leotta, A. Marrella, Interactive Segmentation of User Interface Logs, in: 19th International Conference in Service-Oriented Computing (ICSOC 2021), Springer, 2021, pp. 65–80. doi:10.1007/ 978-3-030-91431-8_5.

[48] A. Marrella, Y. Lespérance, Synthesizing a Library of Process Templates through Partial-Order Planning Algorithms, in: 14th International Conference on Business Process Modeling, Development and Support (BPMDS 2013), Springer, 2013, pp. 277–291. doi:10.1007/978-3-642-38484-4_20.

[49] A. Marrella, M. Mecella, S. Sardiña, Supporting adaptiveness of cyber-physical processes through action-based formalisms, AI Commun. 31 (1) (2018) 47–74. doi:10.3233/AIC-170748.